

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

RESEARCH INTO LANGUAGE CONCEPTS FOR THE MISSION CONTROL CENTER

FINAL REPORT

NASA Grant No. NAG 9-339
SwRI Project No. 05-2768

Prepared by:
Steven W. Dellenback, Ph.D.
Timothy J. Barton
Jeremiah M. Ratner

Prepared for:
NASA
Johnson Space Center
Houston, Texas 77058

June 12, 1990

Approved:



Melvin A. Schrader, Director
Data Systems Department

N91-11392

Unclass
0280759

63/61

CSCL 09B

(NASA-CR-135928) RESEARCH INTO LANGUAGE
CONCEPTS FOR THE MISSION CONTROL CENTER
Final Report (Southwest Research Inst.)
573 p

SOUTHWEST RESEARCH INSTITUTE

POST OFFICE DRAWER 28510 • 6220 CULEBRA ROAD • SAN ANTONIO, TEXAS, USA 78284 • (512) 684-5111 • TELEX 76-7357

June 12, 1990

Ms. Lorraine Rice
NASA - Lyndon B. Johnson Space Center
Houston, Texas 77058

Subject: Delivery of Final Report and MOTIF Comp Builder, NASA Grant 9-339,
Specification Driven Language; SwRI Project 05-2768

Dear Ms. Rice:

Attached is the final report for the Specification Driven Language research grant, NASA Grant 9-339. The final report consists of three separate papers and two appendices. The final report papers describe the research performed during the grant. The first paper describes the Specification Driven Language research. The second paper describes the state of the image processing field and how image processing techniques could be applied toward automating the generation of Comps. The third paper describes the development of a flight certified compiler for Comps.

The first appendix contains source code listings of the original NASA Computation Development Environment (CODE) prototype. The second appendix contains the source code listings of the X Windows MOTIF version of the Comp Builder. A Masscomp 1/4" tape is also included which contains the MOTIF Comp Builder source code in "tar" format.

If you have any questions or comments, please feel free to contact Timothy J. Barton at (512) 522-3540, or Dr. Steven W. Dellenback at (512) 522-3914.

Sincerely,

Susan B. Crumrine

for Melvin A. Schrader
Director
Data Systems Department

MAS:TJB:pam

Attachments

cc: Timothy J. Barton *TJB*

Susan B. Crumrine *SBC*

Steven W. Dellenback

William A. Bayliss

Noel C. Willis (W/O Attachments)

☒ NASA Scientific and Technical Information Facility (W/2 Attachments)



SAN ANTONIO, TEXAS
WITH OFFICES IN HOUSTON, TEXAS, AND WASHINGTON, D. C.

1.0 Introduction

Southwest Research Institute (SwRI) was awarded a grant by NASA-Johnson Space Center (JSC) to perform research in the area of Specification Driven Languages (NASA Grant NAG 9-339). The purpose of the research was to investigate alternative programming techniques/concepts which could be utilized in the software development environments.

This final report is a summary of the work performed under the previously cited grant. In summary, the grant focused on the following areas of research:

- Investigated what techniques NASA and other organizations are using as an alternative form of program development. Primary research included a thorough investigation of three NASA developed languages:
 - Computation Development Environment (CODE),
 - Space Station User Interface Language (UIL), and
 - Ground Operations Aerospace Language (GOAL).
- Performed an in depth analysis and a proof-of-concept prototype implementation of a NASA-JSC defined language (COMputation Development Environment - CODE or Comp Builder) in a UNIX/X Windows environment. The effort involved enhancing the language specification as well as extensive research into alternative user interfaces.
- Performed preliminary investigation into what type of alternative user interfaces could be developed for future software development environments. In the future, the ability to directly generate software programs from engineering drawings will play a major role in NASA's success in rapidly developing and controlling complex systems. A white paper detailing various alternatives was developed and is included in this report.
- Performed preliminary research into how compiler concepts could be applied to a language definition so that a compiler could be developed which would generate flight certified software. Such a tool, with an appropriate user interface, should drastically shorten the program development cycle. A white paper discussing the various attributes that a flight certified compiler should contain was developed and is included in this report.

The results of the research areas described above are contained in the remaining sections of this final report.

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

Specification Driven Language Research

NASA Grant No. NAG 9-339
SwRI Project No. 05-2768

Prepared by:
Steven W. Dellenback, Ph.D.
Timothy J. Barton

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

June 12, 1990

Approved:



for Melvin A. Schrader, Director
Data Systems Science and
Technology Department

Table of Contents

1.0 CODE Language and Implementation Evaluation	1
1.1 CODE Language Evaluation.....	1
2.0 MOAL Language and Environment Research.....	2
2.1 MOAL Language Research	2
2.2 MOAL Environment Research	2
Appendix - A CODE BNF	4
1.0 CODE BNF Rules.....	4
1.1 BNF	4
Appendix - B CODE BNF Evaluation	7
Appendix - C CODE Implementation Evaluation	10
Appendix - D MOAL BNF.....	13
1.0 Notes	13
1.1 BNF	13

1.0 CODE Language and Implementation Evaluation

In order to establish a baseline for the Specification Driven Language grant, a review of existing NASA languages was performed. SwRI investigated the following languages and their constructs:

- Computational Development Environment (CODE)
- Ground Operations Aerospace Language (GOAL)
- Systems Test and Operations Language (STOL)
- TAE Command Language (TCL)
- User Interface Language (UIL) for Space Station Freedom

These languages were investigated to identify currently employed concepts acceptable to the target Specification Driven Language, referred to as the Mission Operations Application Language (MOAL). It was determined that none of the languages currently used or under development would fulfill the requirements of the MCC environment. These languages did however have various constructs that were integrated into the definition of the MOAL.

Initial investigation was focused on the CODE development environment and its language due to its relationship with MCC. CODE was developed by MCC/MOD and had already demonstrated that an "English-like" language tailored to MCC requirements could be developed. SwRI evaluated both the CODE language and its implementation. The evaluation of the CODE language and the evaluation of the CODE implementation are included as appendices to this report.

1.1 CODE Language Evaluation

The initial research of this grant was the evaluation of the CODE language and its constructs. As a means of evaluating the CODE language, SwRI developed a Backus-Naur Form (BNF) representation of the CODE language. The BNF representation permits a more scientific appraisal of the language under study. The BNF allows the implementation of the language to be isolated from the basic language itself. This was important because the work on this grant focused on both the language definition and implementation.

The BNF representation of the CODE language is contained as Appendix A of this report, an evaluation of the language is contained as Appendix B of this report, and an evaluation of the CODE implementation is contained as Appendix C of this report.

2.0 MOAL Language and Environment Research

As CODE's usage is expanded from INCO to more MCC flight controller positions and more NASA programs (Space Station and various payloads), the requirements of the language are changing. SwRI pursued the research and development of both an improved language and an improved development environment. SwRI developed a BNF representation for a language referred to as the Mission Operations Application Language (MOAL). SwRI also developed an X Windows based CODE development environment for evaluation purposes and to provide a platform for definition of the MOAL BNF.

2.1 MOAL Language Research

SwRI developed a BNF representation for the proposed MOAL language. The MOAL language was designed to be an extension to the NASA CODE language. The new language was designed to support all of the MCC flight control positions instead of only INCO. The MOAL was also designed to support the requirements of various payloads, such as TSS. SwRI used the following sources of information to develop the MOAL BNF:

- The MOAL was based primary on the CODE language BNF due to its proven applicability to the MCC environment.
- The MOAL was also based on the requirements of the other MCC positions and the requirements of various payload users. These requirements were compiled by NASA in the Comp Builder and Comp Manager Level B Requirements.
- Research into the UIL and GOAL also determined several aspects of the MOAL BNF.

Appendix D of this report contains the MOAL BNF.

2.2 MOAL Environment Research

SwRI performed research toward defining an improved environment for the MOAL language. This research has been conducted in primarily three areas:

- The first area of research involved the graphical interface of the CODE program. The original CODE program utilized a proprietary Masscomp graphics interface. For evaluation purposes, this interface was converted to an MIT X Windows graphics interface. The first version of the X Windows Comp Builder utilized the Hewlett-Packard (HP) widget set and served as a baseline for implementation of an improved user interface. The X Windows prototype also included several performance enhancements to the basic structure of the Comp Builder software. The X Windows prototype was well received by NASA and served as the basis for the WEX 2.5 version of the Comp Builder.

The second version of the X Windows Comp Builder utilized the Open Software Foundation (OSF) MOTIF widget set. The MOTIF version of the Comp Builder served as a test-bed platform for the development of the MOAL BNF and several enhancements to the user interface.

- A preliminary investigation into alternate input techniques for the MOAL environment was conducted. This investigation resulted in the Image Processing paper included in a later section of this document.
- A preliminary investigation into a flight certified environment was also conducted. This investigation resulted in the Flight Certified Compiler paper included in a later section of this document.

1.0 CODE BNF Rules:

$$\vdash = \{ \} ..$$

The terminal: PI represents the arithmetic constant 3.159.

```
comment ::= /* { <alphanumeric> } */
```

action_stmtnt	::= <set_statement> <print_stmtnt>
alphanumeric	::= <letter> <digit> <id_char>
comp	::= <header> <statement_list>
defined_func	::= cos acos sin asin tan atan exp log sqrt power
digit	::= 0..9
factor	::= <variable> (<simple_expr>) <func_designator> - <factor>
float	::= <float_no_zero> { <digit> . } (1 decimal point max)
float_no_zero	::= 1..9 .
func_designator	::= <defined_func> (<factor> { , <factor> }) <identifier> (<factor> { , <factor> })

group	::= <comp> { <comp> }
header	::= /* <create_str> <author_str> <gname_str><cname_str> <purpose_str> */
hex_char	::= A..F a..f
id_char	::= _ -
identifier	::= <letter> { <alphanumeric> }
if_statement	::= if <rel_expression> then <statement_list> endif if <rel_expression> then <statement_list> else <statement_else> { <statement_list> } endif
integer	::= <no_lead_zero> { <digit> <hex_char> }
letter	::= A..Z a..z
local	::= <identifier>
MSID	::= <identifier>
no_lead_zero	::= 1..9 <hex_char>
number	::= <integer> <float>
operator	::= + - or * / and bitAnd bitOr bitXor shiftL shiftR
print_statement	::= print1 <quoted_string> print2 <quoted_string> print3 <quoted_string> print4 <quoted_string> print5 <quoted_string>
quoted_string	::= " { <alphanumeric> } "

rel_expression ::= <simple_expr> <rel_operator> <simple_expr>

rel_operator ::= < | > | <> | = | >= | <=

set_statement ::= set <set_var> = <factor>

set_var ::= <signal> | <local>

sign ::= + | -

signal ::= <identifier>

simple_expr ::= <sign> <factor> | <factor> <operator>
<simple_expr>

statement ::= <if_statement> | <action_stmtnt>

statement_else ::= <action_stmtnt> | <variable> = <factor>

statement_list ::= <statement> { <statement> }

variable ::= <set_var> | <MSID> | <number> | <quoted_string> | PI

author_str ::= Author: { <alphanumerics> }

cname_str ::= Comp Name: xxxxxxxx

create_str ::= Creation Time: xxxx:xx:xx:xx

gname_str ::= Group Name: xxxxxxxx

purpose_str ::= Purpose: { <alphanumerics> }

Appendix - B CODE BNF Evaluation

The CODE grammar is more compact than a traditional High Order Language (HOL) due to the fact that CODE has no formal data typing within the language (the CODE environment maintains a table of data types), a limited number of statements and a slightly restricted relational expression capability.

The representation of CODE in a BNF form facilitated the following evaluation:

- CODE was designed to meet the needs of Mission Operations Directorate (MOD) INCO flight controllers. This required only the following three statements:
 - Conditional control statement - if
 - Assignment statement - set
 - Output statement - print

CODE users have already requested additional conditional and assignment statements.

- The IF statement allows invalid statements to be added after the ELSE. There are two sets of statements available when constructing the ELSE portion of an IF statement:

```
if rel_expr
then
    { group1 }
else
    group2
    { group1 }
endif
```

The first statement following the ELSE allows the user to select an MSID, SIGNAL, or LOCAL, and then assign a value to these variables. These selections should not be valid unless they are contained in a SET statement. The ELSE portion of the IF statement should allow only the same statements as the THEN portion.

- The CODE language allows a value to be assigned to an MSID. This is not a valid operation, as MSIDs are values which are read from data acquisition and are not modifiable by the CODE software.
- A user function must contain one argument. CODE forces an argument to be supplied in user function calls. It is standard programming practice in HOL's to allow function calls with no arguments. The CODE language should allow these types of function calls.

- The CODE program does not allow a SET statement to be added to a Comp after a SET statement is added which contains a user function call on the Right Hand Side (RHS). Two or more SET statements can be added to a Comp as long as none of them contains a user function call. The CODE program should allow multiple SET statements to be added which contain user function calls. Example:

```
set var1 = 100
set var2 = userFunc1( userFunc2( 5 ) )
/* at this point, CODE will not allow another SET statement to be */
/* added to the Comp, but will allow a PRINT or IF statement      */
```

The irregularity described above always occurs when the SET statement is the first statement in a Comp.

- Traditionally, in both C and PASCAL, expressions are allowed to evaluate to a simple expression. The CODE language only allows expressions to be evaluated to two simple expressions separated by a relational operator. This restricts the type of expressions that can be constructed. The CODE language designers may have intentionally designed CODE in this manner. If a relational operator is always required, the user is not required to know that 0 evaluates to FALSE, while any other value evaluates to TRUE.

Traditional HOL

```
expression ::= <simple_expr> | <simple_expr>
              <rel_operator> <simple_expr>
```

CODE

```
rel_expression ::= <simple_expr> <rel_operator> <simple_expr>
```

This feature of the CODE language is not standard when compared with traditional HOL's, but is appropriate for the non-programmer audience served by CODE.

- CODE allows the RHS of a SET statement to be a <factor>. Traditional HOL's allow an <expression> non-terminal on the RHS of an assignment statement. This permits greater flexibility in the assignment of data values to variables.

Traditional HOL

```
set_statement ::= set <set_var> = <expression>
```

CODE

```
set_statement ::= set <set_var> = <factor>
```

- The PRINT statement should only allow WEX application level messages to be output. The current CODE will allow the running comp to generate host level messages. The WEX version of CODE should restrict the type of messages generated. This issue is environment dependent, in this case the WEX environment determines the valid set of PRINT statements. Even though the valid PRINT statements are environment dependent, the language BNF has determined that all PRINT statements are valid at any time. The CODE language should be modified to only allow those PRINT statements that are valid for WEX applications.
- The CODE BNF should allow the formal definition and construction of user functions. The CODE language and environment do not allow the definition of user functions. User functions must be constructed outside the CODE environment. The CODE language should include the constructs necessary to define and reference user functions built within the CODE environment.
- The CODE documentation should either define the order of precedence for expression evaluation or inform the user of the dependence on the host C compiler. CODE's order of precedence of expression evaluation is directly tied to the C compiler used to install the Comps. CODE will allow the following Comp statement to be generated:

if $PI * \tan(1) + PI > 1$

This statement should be valid, but the order of expression evaluation is unknown without understanding how the Comp is translated into the resulting 'C' language program.

- The CODE language allows the negation of a text string. Example:

set Var1 = userFunc1(- "text string" + 1)

- CODE allows integers and floating point numbers to be 30 characters long. This allows the user to enter numbers which are too large to be represented using standard integer arithmetic. This is both a CODE language and CODE implementation issue.
- The CODE language allows the user to build Comps which contain the assignment and evaluation of mixed type expressions. These expressions may be cast by the C compiler or may cause fatal errors during the installation process. Example:

set Var1 = .5 + "text string" + 10

This is another case of an issue which should be addressed by both the CODE language and CODE implementation.

Appendix - C CODE Implementation Evaluation

After the evaluation of the CODE language, the CODE program was evaluated from a software design aspect. SwRI evaluated CODE's stability, portability, and overall system design.

The following is a list of problems and/or deficiencies identified in the NASA/MOD version of CODE. Several of the following items relate to the hardware differences between the MASSCOMP systems used by the CODE developers and SwRI's MASSCOMP. These items are irrelevant in a strictly homogeneous hardware environment, but such an environment should not be assumed due to the rapid advancement of workstation hardware development. A homogeneous hardware environment also adds requirements to the various contractors which develop and support NASA software. An X Windows version of the Comp Builder was requested by several NASA contractors because Masscomp workstations were not available to them, but X Window capable Sun workstations were available.

- The CODE program needs to determine the resolution of the monitor before displaying the token strings. CODE currently uses a font which is larger than the boxes allocated for the strings. The font can be changed to a smaller font which fits nicely in the boxes, but the portions of the text which exceeded the size of the boxes are not cleaned up and remain on the screen until CODE is exited.
- The user interface should be improved such that the user is not required to continue switching back and forth between the mouse and the keyboard. A function needs to be added so the user can mouse an area to continue processing instead of hitting the RETURN key. This enhancement was made in the X Windows versions of the Comp Builder.
- In the case that the RTDS environment variable is not defined, CODE currently exits, leaving the graphics on the screen. CODE should restore the screen and remove the graphics. CODE should also allow the user to enter the RTDS path if the user so desires. This enhancement was made in the X Windows versions of the Comp Builder.
- The CODE program currently uses 5 meg of static data space for tables and arrays that it maintains internally. This amount should be reduced through the use of dynamic memory allocation. The following data structures are examples of structures that should be dynamically allocated:

SignalTable	- 800 structures of 3081+ bytes =	2,464,800
MSIDTable	- 4000 structures of 72 bytes =	288,000
GroupVars	- 500 structures of 100+ bytes =	50,000
Comp	- the Comp string	20,000

- The READ process tries to rm (delete) a file using the file pointer instead of the ASCII name of the file. This will produce undetermined and potentially disastrous results. The source file archive.c contains the incorrect code:

```
tmpfile = fopen( ... );  
system( "rm tmpfile 2>>/tmp/code.err" );
```

- The READ process uses the variable "found" before it is initialized. This could produce undetermined results should the Masscomp compiler change or if the CODE program is ported to another version of the Masscomp operating system or to another hardware platform, such as a Sun.
- The CODE program contains a bug in the software that performs the DELETE function. The following steps will cause the CODE program to abort:

Retrieve a Comp

Delete several tokens from the end of the Comp using DELETE

Save the Comp

Retrieve the Comp

The next time the Comp is retrieved, the CODE program will abort due to a memory Segmentation Fault and a "core" file will be created.

- The DELETE function should be modified to let the user delete any token. The current DELETE function only allows the user to delete the last token in the Comp. If the user wanted to delete the first token in the Comp, the entire Comp would have to be deleted one token at a time or the user could use the VI editor to delete the token.
- CODE allows integers and floating point numbers to be 30 characters long. This allows the user to enter numbers which are too large to be represented using standard integer arithmetic. This issue was also identified in the language evaluation section of this report. Both the CODE language and the CODE implementation should address this issue.
- Using some Masscomp configurations, the VI window does not size itself correctly within the Work Area. VI is unusable as the CODE editor, the user must use VED to edit the Comp string.
- The date calculation performed in get_header.c does not compute the date correctly. Subsequently, the date recorded in the header of all Comps is incorrect.
- Several path specifiers should be corrected. There are currently paths which are built containing multiple forward slashes. These paths could cause portability problems should a new version of the Masscomp operating system not support this type of path specifier. The following line in init_code.c is an example:

```
UserFuncsLib, "/home//rtds/..."
```

- In init_code.c, if the MSID Table is not found, the loop control variable MSIDTble is not initialized. During the install() routine, MSIDTble is used to access arrays in memory. This uninitialized variable could cause the CODE program to access

invalid sections of memory producing undetermined results. MSIDTble should be explicitly initialized for portability.

- In `init_gp.c`, the call to `signal()` which is to trap Control-C is commented out. A Control-C entered in the prompt area will abort the program leaving the CODE graphics on the screen. The Control-C signal handler should be implemented.
- It is possible to build comps using the tokens available on the screen (it is not necessary to use VI or VED) that when compiled produce C code, which may be invalid. The produced C code makes several calls to `fltnsg_issue()` before the `fltnsg_init()` call.
- Using a Masscomp 5500, the tokens in the bottom right corner of the MATH token box were not mouseable during HELP mode. If the tokens are selected, the CODE banner is displayed, and the FONT is changed. This an example of hardware dependency that should be eliminated where possible.
- The documentation needs to be updated to include all the tokens available during HELP mode. There are currently as many undocumented tokens as there are documented tokens. A user friendly program should have all the help messages defined.
- The large font may not be used during HELP mode, as the first lines of every screen are scrolled out of the top of the Work Area window.
- The comp string display routine does not allow the user to view an entire comp if it is larger than the Work Area window. Only the last section of the comp (that portion that fits within the Work Area window) is visible. There is no means to view the top of the comp except for VI or VED.
- The `openFile()` routine needs to let the user see the message longer (ideally a mouseable continue box). Currently, the message disappears from the screen too fast for the user to read.
- CODE does not type check the arguments to user functions to ensure they match the required parameters. Passing the wrong parameters to a user function at best case will cause the comp to abort, in worst case will cause the comp to continue execution with side effects; possibly undetected side effects.
- The UNIX system program lint generated several pages of irregularities when it examined the CODE program C source code. The following errors are typical of the numerous anomalies discovered by lint:

Redefinition of variable data types

Usage of uninitialized variables

Declaration of unused variables

Incorrect number of arguments passed to functions

Appendix - D MOAL BNF

1.0 Notes:

The following symbols are meta-symbols belonging to the BNF formalism:

::= | { } ..

The non-terminal: <group> is the start symbol.

See Appendix E of the Comp Builder and Comp Manager Level B Requirements (JSC-23459) for the Naming Conventions and limitations of the non-terminal: <identifier>.

See Appendix C of the Comp Builder and Comp Manager Level B Requirements (JSC-23459) for the data representation and precision limits of the referenced data types.

The statement terminator is the newline character (carriage return). An exception is <expressions>, which may require multiple source lines.

1.1 BNF

add_operator ::= + | - | or | xor

alphanumeric ::= <digit> | <letter> | <special_char>

char ::= ' A..F | a..f '

comp ::= <comp_header> { <parm_list> } { <const_list> }
{ <var_list> } <statement_list>

const_decl ::= const <data_type> <identifier> = <const_def>

const_def ::= <unsigned_num> | <sign> <unsigned_num> | <char> |
<identifier> | <sign> <identifier> | <string> |
<matrix> | <matrix_id>

const_list ::= <const_decl> { <const_decl> }

data_type ::= integer | unsigned | float | char | double |

matrix | short

defined_func	:= cos acos sin asin tan atan cuber log cosh exp sinh tanh sqr log10 start terminate
digit	:= 0..9
expression	:= <simple_expr> <rel_operator> <simple_expr> <simple_expr>
factor	:= not <factor> <function_id> (<expression>) <unsigned_num> <identifier>
function	:= <func_header> { <parm_list> } <return_decl> { <const_list> } { <var_list> } <statement_list> <return>
function_id	:= <defined_func> ({ <parms> }) <identifier> ({ <parms> })
group	:= <comp> { <comp> <function> <procedure> }
hex	:= A..F a..f <digit>
identifier	:= <letter> { <alphanumeric> }
if_stmnt	:= if <expression> then <statement_list> endif if <expression> then <statement_list> else <statement_list> endif
letter	:= A..Z a..z
local_id	:= <identifier>
matrix	:= [<value> { , <value> }]
matrix_id	:= <identifier>

sign	::= + -
simple_expr	::= <simple_expr> <add_operator> <term> <term> <sign> <term>
special_char	::= _ -
statement	::= <print_stmnt> <set_statement> <if_stmnt> <while_stmnt> <pause_stmnt> <proc_id>
statement_list	::= <statement> { <statement> }
string	::= " { <alphanumeric> } "
term	::= <factor> <term> <mul_operator> <factor>
unsigned_hex	::= 0x <hex> { <hex> } 0X <hex> { <hex> }
unsigned_int	::= <digit> { <digit> }
unsigned_num	::= <unsigned_int> <unsigned_real> <unsigned_hex>
unsigned_real	::= <unsigned_int> . <unsigned_int>
variable	::= <set_var> <MSID_id>
var_decl	::= var <data_type> <identifier>
var_list	::= <var_decl> { <var_decl> }
value	::= <identifier> <sign> <identifier> <unsigned_num> <sign> <unsigned_num>
W/S_parm_id	::= <identifier>

while_stmt ::= while <expression> loop <statement_list> endloop

comp_header ::= Position Name:
Group Name:
Comp Name:
Author:
Creation Date:
Last Update:
Purpose:

func_header ::= Function Name:
Author:
Creation Date:
Last Update:
Purpose:

proc_header ::= Procedure Name:
Author:
Creation Date:
Last Update:
Purpose:

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

IMAGE PROCESSING

NASA Grant No. NAG 9-339
SwRI Project No. 05-2768

Prepared by:
Timothy J. Barton
Bradley Adams

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

June 12, 1990

Approved:

Susan B. Crumrine
for Melvin A. Schrader, Director
Data Systems Science and
Technology Department

Table of Contents

1.0 Introduction	1
2.0 Definition of the Fields of Image Scanning	2
2.1 Image Digitization/Optical Scanning.....	2
2.2 Image Processing	2
2.3 Image Recognition/Pattern Recognition	2
2.4 The Complete Package	2
3.0 Image Digitization	3
3.1 Image Digitization/Optical Scanning.....	3
3.2 Origins	3
3.3 How Scanners Work	3
3.4 Scanner Designs.....	4
3.5 Commercially Available Scanners.....	4
3.5.1 Commercially Available Scanners for the PC	5
3.5.1.1 PC Bundled Software.....	5
3.5.2 Commercially Available Scanners for UNIX Workstations.....	5
3.6 The Future of Optical Scanners	6
4.0 Image Processing	7
4.1 Classical Image Processing Algorithms.....	7
4.2 Noise Reduction.....	7
4.2.1 Edge Enhancement/Detection.....	7
4.2.2 Dithering	8
4.2.3 Gray-scaling.....	8
4.3 Graphics Packages	8
4.3.1 Graphics Packages for the PC.....	9
4.3.2 Graphics Packages for the Workstation.....	9
4.3.3 Standard Graphics File Formats.....	10
4.3.3.1 Tagged Image File Format (TIFF).....	10
4.3.3.2 PostScript Format (PSF)	10
4.4 Computer Aided Design (CAD)	11
4.4.1 CAD Goals.....	11
4.4.2 CAD Packages for the PC.....	11
4.4.3 CAD Packages for the Workstation.....	12
4.4.4 Standard CAD File Formats	12
4.4.4.1 EDIF/IGES.....	12
4.4.4.2 DXF	13
5.0 Image Recognition.....	14
5.1 Optical Character Recognition (OCR).....	14
5.1.1 Origins	14
5.1.2 How it Works.....	14

5.1.3 Matrix Matching	15
5.1.4 Feature Extraction	15
5.1.5 Hybrid Methods	16
5.1.6 Commercially Available OCR	16
5.1.6.1 OCR for the PC	16
5.1.6.2 OCR for the Workstation	16
5.2 CAD Overlaying	16
5.2.1 How it Works	17
5.2.2 Commercially Available CAD Overlay	17
5.3 Raster-to-Vector Conversion	17
5.3.1 Raster-to-Vector Conversion Packages for the PC	18
6.0 Image Processing/Image Recognition Research	19
6.1 Research in Graphics/Text Separation	19
6.2 Research in Reading Engineering Designs	20
7.0 Possible Directions Toward A Solution	24
8.0 Referenced Vendors	25
9.0 Bibliography	26

1.0 Introduction

NASA-JSC currently maintains engineering diagrams and schematic drawings of many of the subsystems contained within the shuttle. These diagrams and drawings are often converted into a drawing containing logic symbols and text which describe an algorithm for monitoring the status of the subsystem. Figure 1-1 contains an example of a logic drawing which describes an algorithm.

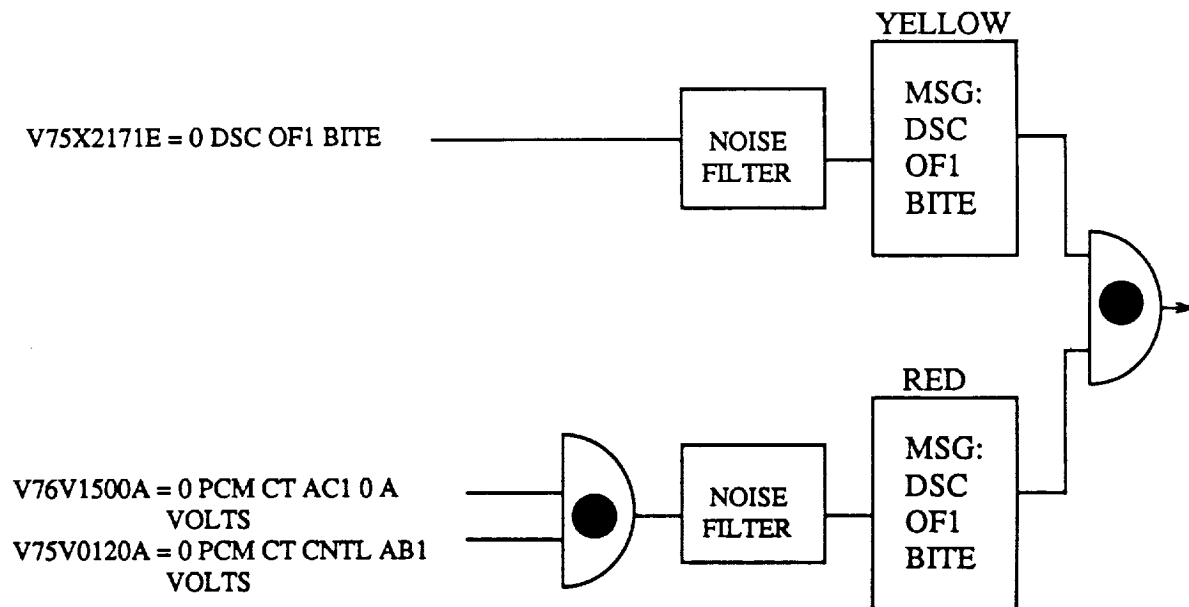


Figure 1-1 Example Logic Drawing

The algorithms described by the logic drawings will perform range checking of data points to determine the operational stability of the subsystem. These drawings are interpreted by a human and implemented using programming language type constructs. A more efficient method of software generation would be to provide an automatic process which could "scan" the schematic drawing, as would a human, and develop a program with minimal (or no) programmer involvement.

Implementation of such a system would require an orchestration of systems from the following technologies and topics: Image Digitization, Image Processing, Pattern/Image Recognition, and Automatic Code Generation. To gain understanding of the current capabilities of these technologies, this paper will address each area individually. The intention of this breakdown is to provide insight into the current state of the related technologies, what tools are available in each, and where the fields are headed. Special emphasis will be given to viewing the individual technologies in light of the problem at hand.

2.0 Definition of the Fields of Image Scanning

A widely held misconception is the idea that all scanners (hardware and software combined) automatically interpret the documents they digitize. This is not the case. For scanners set up to perform interpretation of text alone (Optical Character Recognition), this is true, but for a mix of text and graphics, this is not the case. Another widely held misconception is the idea that the hardware issues present more of a challenge than the supporting software. Presently, this is not the case. It is important to first understand the various components of document scanning and the terms that are used to identify them.

2.1 Image Digitization/Optical Scanning

Image Digitization identifies the conversion of a physical image to a digital image stored within a computer. Image Digitization is primarily a hardware intensive function. No attempt is made to enhance the quality of the digital representation or to interpret the content of the digital image. This paper will use the term scanner to identify the hardware that generates a digital image and the term scanning to identify the process of generating a digital image.

2.2 Image Processing

Image processing identifies the manipulation of a digital image. Edge enhancement, noise filtering, and image cut and paste operations are examples of image processing techniques.

2.3 Image Recognition/Pattern Recognition

Image recognition identifies the processing of a digital image to determine the logical contents of the image. OCR is an example of image recognition. OCR software attempts to determine the actual characters of a digital image.

2.4 The Complete Package

Image scanning often involves the integration of all three of the aforementioned disciplines. First, the image is scanned (digitized). During digitization, the image may be processed using edge enhancement and noise reduction techniques. Depending on the application, recognition of the image may then be performed. To provide a tool for the automatic generation of algorithms from logic diagrams, all three disciplines will be required.

3.0 Image Digitization

The first step in the scanning of a document, say the logic diagram in Figure 1-1, is the digitization of the image. Image Digitization can also be referred to as "scanning." Image digitization is a more cumbersome term, but it more accurately describes the process which is performed. The term "scanning" is often used as a general term for the combination of image digitization, image processing, and image recognition.

3.1 Image Digitization/Optical Scanning

The goal of optical scanning is to convert a physical image into a digitized image, whether that image be a page of text, a picture, a combination of text and graphics (like a logic diagram), or a three dimensional object. A digitized image is a binary file of ones and zeros stored within a computer that represents the physical image. Software can use this binary information to reproduce the image on a computer screen, on a printer, or on a remote fax receiver.

3.2 Origins

The development of scanners has its origins several years ago in Japan where scanners were applied to fax. Since Japanese characters are essentially graphical in nature, fax is the easiest way to transfer documents.

On the other hand, scanner related software development in the United States has evolved mainly at the shoulder of word processing and desktop publishing demands. With the large volume of paper documents that need to be processed, the ability to scan in text from a physical sheet of paper to a word processing format where it can be manipulated has received a lot of attention. This process is commonly called optical character recognition (OCR). Likewise, the ability to digitize and process pictures that could be incorporated into documents has also been a priority topic (cut and paste of document graphics). As we shall see later, however, of the two scanner software products that developed around the word processing industry, only optical character recognition requires software interpretation or recognition of symbols within the digitized image. Image scanning, as it relates to pictures within word processing documents, only manipulates the bits of the image, not the symbols contained therein.

3.3 How Scanners Work

All non-color scanner technology is based on the fact that black pigment absorbs white light, and white pigment reflects it. A similar principle is used in color scanners that involves colored lights, filters, and pigments. The scanner bounces light off the image that is to be captured. If the light reflects off the image, then it reaches photoelectric cells called Charge Coupled Devices (CCD) which measure the intensity of the reflected light within their cell boundaries. The size of the CCDs and how closely they are packed together determines the resolution of the scanner. Each CCD outputs a given voltage for a given intensity of reflected light. Zero voltage represents no reflected light detected and would indicate that portion of the image is black.

A controller in the scanner takes the voltage measurement from each of the scanner's CCDs and converts it into a whole number which is stored in the computer. Because the CCDs can output only increments of the measured voltage, the true intensity is rounded to a closely cor-

responding voltage level. This is how gray-scale information from the physical image is lost in the digitized image. The controller uses the numbers collected from the CCDs to create an image bitmap file that saves one number for each pixel in the image for later use by a graphics package or other software. The graphics package, for example, would take the values contained in the bitmap file and approximate the image by lighting pixels on the screen according to the gray-scale values.

Color scanners work using the same basic principles, but color scanners use different lighting. The CCDs in color scanners use color filters, and the bitmap files contain intensity values for each pixel for each primary color. This also means the image files are three times larger than black and white image files of the same type.

3.4 Scanner Designs

There are currently four major, different configurations of scanners:

- Flatbed
- Roller-feed
- Copyboard
- Hand-held

Flatbed scanners comprise the majority of the scanners available in the market currently. Of the 21 scanners reviewed by PC Magazine in the March 28, 1989 issue, 16 were flatbed scanners. Flatbed scanners are versatile, and they utilize a traditional design borrowed from small photocopiers. Flatbed scanners are preferred for ease of use and for their ability to scan bound documents such as books and magazines.

Roller-feed scanners employ the flatbed scanner design with an additional sheet feeder mechanism. Manufacturers often offer a sheet feeder mechanism for their flatbed scanners.

Copyboard scanners look very much like a photograph enlarger or overhead projector. Copyboard scanners consist of a flat platform with the scanning device raised approximately 12" above the copyboard. Copyboard scanners are primarily designed for scanning 3-D objects, such as circuit boards, but can be used to scan flat documents as well. Copyboard scanners often require special lighting due to the distance between the scanner sensors and the subject matter.

Hand-held scanners are a relatively new design. Hand-held scanners, such as the Logitech ScanMan, often utilize a 4" wide scanning window; however, Mitsubishi Electronics offers a full 8.5" wide hand-held scanner. They have become very popular because they are inexpensive, and they have proven very useful for importing images into desktop publishing and word processing software.

3.5 Commercially Available Scanners

Scanners are now available from a wide variety of vendors for various hardware platforms. Scanners are often available from the same vendors that produce photocopiers and laser printers due to the similarities of the technologies. The personal computer (PC) market accounts for the majority of scanners purchased currently in the United States due to its large number of users and the type of applications commonly used on PCs. NASA-JSC currently

uses UNIX based MassComp workstations to implement its algorithms, so a survey of the workstation market is also included.

3.5.1 Commercially Available Scanners for the PC

The use of scanners as an input peripheral in the personal computer arena has progressed significantly in the last five years. There are currently more than 150 different products. These products range from the \$169.00 Logitech hand-held scanner to the \$15,950.00 Kurzweil Imaging System. According to the market research firm Dataquest, scanner sales have increased an average of 250 percent annually from 1984 to 1987. Another 200 percent increase was projected for 1988. In just three years, scanner manufacturers have doubled resolution, added gray-scale capability, and standardized graphics file formats.

The scanners reviewed in the May 1989 issue of the DataPro Research Bulletin exemplify the diversity of image scanners available. Three different scanners from three different vendors were reviewed, each representing a different segment of the PC scanner market. The first scanner reviewed was the Hewlett-Packard ScanJet Plus, an enhanced version of the popular HP ScanJet. The ScanJet Plus is relatively inexpensive, retailing at \$1,595 plus \$595 for a PC interface kit. It is supplied with very powerful software including OCR software which allows generation of WordPerfect files from scanned documents. It is very easy to use, and it has an extensive range of gray scale. The Microteck Scanners were reviewed next. Ranging from \$995 on their low-end model to \$3,995 on their high end, they have an exceptional image quality, and they work with many microcomputer systems, including Sun workstations. Microteck also bundles advanced SuperPaint graphics software with their scanners. Finally, the Xerox Pro Imager, with its \$10,000 price tag, provides superior performance with 1,200 dpi resolution. The ProImager's software also allows image enhancement which goes beyond the average capabilities of graphics packages that may be bundled with scanners.

3.5.1.1 PC Bundled Software

A discussion on image manipulation and OCR is better suited for the section in this paper that addresses image processing and image recognition, so a more thorough explanation of the state of those fields is left until then. However, it is worth mentioning that an important trend in the PC arena is the marriage of third party software, such as OCR, with OEM scanner hardware and image manipulation (graphics package) software. For example, Cannon and Chinon supply OCR System's ReadRight with their scanners. Again, it is the word processing and desktop publishing industries that have carried scanner and scanner-related software with it, thus the most common areas of software development have been in OCR and graphics manipulation. Twenty-five percent of scanners sold currently have OCR capability. The majority of existing scanners are used for graphics capture and manipulation.

3.5.2 Commercially Available Scanners for UNIX Workstations

The current process used by NASA-JSC to build its algorithms utilizes UNIX based MassComp workstations. Due to its smaller segment of the computer market and more complex and diverse hardware interfaces, fewer optical scanners are available for the UNIX workstations. MassComp (Concurrent Computer Corp.) currently does not offer a scanner or even list a third party vendor in their Solutions catalog that produces a scanner targeted for the

workstation market. Microtek Lab Inc. is currently the only vendor that offers a scanner for the very popular Sun Microsystems workstations.

Often, a scanner whose target market is PCs, will be connected to a workstation through a similar interface. The NASA-JSC Mission Operations Support Lab (MOSL) is currently using a Howtek Scanner connected to a MassComp. The Howtek scanner is targeted toward the much larger PC market but can be connected to any computer that utilizes a GP-IB interface. The GP-IB interface is not very common on workstations, though, and the software which is typically bundled with the Howtek scanners cannot be used on the workstation. NASA-JSC has had to write their own software to allow the images from the Howtek scanner to be displayed on the workstation. Compare this with the PC market where just about every scanner includes (or has available) an interface and supporting software bundled with the scanner.

Microtek Lab Inc., as mentioned previously, does offer a scanner and software targeted for the Sun Microsystems UNIX workstations. Microtek offers two different models of scanners and bundles graphics manipulation software for use in the SunView window environment. Microtek Lab is at this writing the only vendor available even to Sun users, the largest workstation market segment. Compare this with the PC market, where more than 150 different products are available.

3.6 The Future of Optical Scanners

Scanner developers will be trying to get higher resolution, better color, etc. for a long while yet. In addition, many scanner manufacturers are converting from TIFF output formats to true PostScript image output files. See Section 4.3.3 for a discussion on the various file formats. Scanning is heading toward a future of unification with other computer input and output peripherals, as well as with other image processors that are not computer peripherals. Manufacturers and users would like to see one combination machine that acts as copier, scanner, laser printer, and fax.

In the UNIX based workstation arena, more scanners and supporting software should become available as the workstation market continues to grow. As users migrate their applications from PCs to workstations in search of more processing power, the need for scanners and related software will increase. The wide acceptance of X Windows provides a non-proprietary graphics environment for scanner software development.

4.0 Image Processing

Image processing refers to the processing of an existing digital image. In his book Robot Vision, Berthold Klaus Paul Horn defines image processing as "the generation of new images from existing images." The definition of image processing as used throughout this paper, with the exception of the title of this paper, is the more traditional definition of image processing. Image processing can also be thought of as the processes performed between image digitization and image recognition.

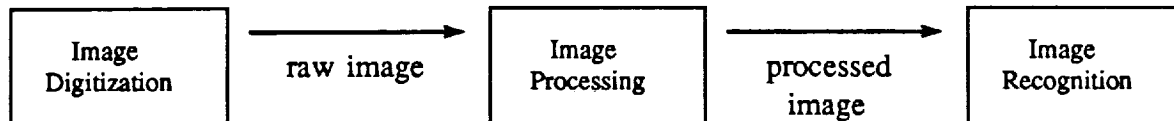


Figure 4-1 Image Flow

Image processing is also used as a broader term to describe the entire process of scanning an image through image recognition.

4.1 Classical Image Processing Algorithms

Classical image processing deals with the raw digital images produced by scanners and video cameras. Image processing in these arenas is typically geared toward improving the quality of the digital image to provide a better image for the subsequent image recognition or image generation (screen display or printing) processes. Many of the early classical image processing techniques were derived or evolved from linear systems theory. Most of the work in classical image processing has been done as a preprocessing step for machine vision and pattern recognition. In recent years, PC scanners and their related software have borrowed from classical image processing to improve the quality of their images for OCR and graphics packages.

4.2 Noise Reduction

Images input by scanners are naturally susceptible to many types of noise. Noise within a gray-scale image can be easily removed by a form of interpolation or by Fourier transforms when the noise is isolated and has nonnoise neighboring points. This type of noise is referred to as "salt-and-pepper" noise. Other types of noise, even if they affect only a small percentage of the total image, are very hard to correct inconspicuously. A great deal of research is being conducted currently to determine the best numerical calculations to remove noise from gray-scale images. Research is investigating simple filtering and threshold algorithms and complex Fourier transforms to remove noise from images.

4.2.1 Edge Enhancement/Detection

Edge detection and enhancement are an important step in the processing of an image. In the case of OCR, it is important that the edges of each character are detectable so the character can be identified. In the case of image recognition of a logic drawing, it is important that the

shape of the logic symbols and the lines connecting the symbols can be identified. Noise reduction is an important first step before edge enhancement/detection can be performed. It is important that the noise reduction processes reduce noise around edge boundaries, especially short length edges. Shorter length edge segments must have a higher, cleaner contrast than longer length segments to be detected. This fact is illustrated by the segment lengths imposed by the Toshiba engineering diagram reader (See discussion in Section 6.2)

Edge enhancement/detection is performed by detecting a border between two regions, each of which has approximately uniform brightness. According to Horn, most work has been done on the edge enhancement process with less work being done on actual edge detection. As image recognition progresses, edge detection becomes critical so the grouping of line segments can be performed to identify shapes.

4.2.2 Dithering

Standard black-and-white photographs may contain more than one million shades of gray. Current PC-related scanner and software technology does not provide for the processing or output of this many shades of gray. As a result, dithering and gray-scaling are two methods which are used to improve the handling of continuous-tone images.

Dithering is currently the most popular method of improving continuous-tone images in PC image processing. As the gray tones are measured by the scanner, they are converted into various bitmap patterns. A high gray tone value is represented by a very dense bitmap pattern, and low gray tone is represented by a less dense bitmap pattern. Dithered images are editable, which means the bitmap for a region of the image can be altered. Dithered images are difficult to scale and are also difficult to vary in resolution. Scaling and image resolution are often modified for printing, so this is a disadvantage of dithering. Dithered images are often stored in PCX, PIC, or IMG formats.

4.2.3 Gray-scaling

Gray-scaling is another method used in PC image processing to store continuous-tone images. Gray-scaling is relatively new to the PC arena and is not used as widely as dithering. As previously described, dithering converts the gray tone value returned from the scanner into a bitmap pattern. Gray-scaling, on the other hand, keeps the gray tone value in the image file to be used later by the display software or hardware (printer or graphics display). The quality of the displayed image is determined by the range of grays available to the output device.

The majority of existing PC hardware and software does not produce gray-scale quality output. Laser printer manufacturers and software developers are working feverishly to develop gray-scale quality products. Big name corporations such as Intel, Hewlett-Packard, and Microtek have either announced gray-scale quality products or will shortly.

4.3 Graphics Packages

Graphics packages provide the user with the ability to create and manipulate images. For example, a typical graphics package would provide the user with the ability to create pictures consisting of lines, shapes, surfaces, colors, and text, as well as the ability to bring in a scanned image stored in a standard scanner output format. With the image loaded into the graphics package, the user can trim away parts of the picture, or cut and paste portions of it

as desired into his word processing and desktop publishing documents. Some more advanced graphics packages even allow the user to enhance the image by changing color content or by similar functions.

Applications that have had less of an impact on off-the-shelf image processing software, but more of an impact on the advancement of the field of study, are applications applied to special fields such as astronomy, medicine, geology, and defense. For example, computer x-ray tomography (CT) scanner images are processed by advanced medical applications to extract information about body function and to identify unusual growths without using surgery. To do this, the graphics software enhances images, rearranges picture parts, enhances color separations, or improves shading.

4.3.1 Graphics Packages for the PC

There is an endless number of PC paint and graphics packages that have been developed during the micro-computer revolution. Many of the packages originated mostly for entertainment and drawing practice. However, the demands of word processing and desktop publishing pushed the development of more advanced and complete systems. These more advanced systems combine text, drafting capabilities, freehand modes, and image processing functions into all-purpose packages that can be used by artists, illustrators, image processors, and desktop publishers.

Without question, the leaders in the PC graphics world are Adobe Corporation's Illustrator and Aldus Corporation's Freehand. Neither could be classified as a dominant number one, but together they seem to hold that position. Both are list priced at around \$695. These graphics packages are the deluxe models because they can import and export a large number of different image formats, including true PostScript. They allow freehand drawing with many different colors and brushes. The packages give drafting capability by manipulating squares, lines, arrows, and all shapes as separate entities that can be moved, resized, overlaid, etc. In addition, text of all sizes and fonts can be mixed with the graphics and freehand drawings. Both are available on the Apple Macintosh computers and on the IBM PCs.

4.3.2 Graphics Packages for the Workstation

MassComp (Concurrent Computer Corp.) workstations have a fair amount of graphics software that is intended to be used as graphics routine libraries that are used to build other application programs. In the most recent release of their Solutions Catalog, no user interactive graphics package is mentioned. In summary, there is no software on the MassComp that will allow a user to interactively build or manipulate image files, while there are a great deal of tools or libraries that might be used by programmers to build such a system.

Sun Microsystems lists in its most recent issue of the Catalyst Catalog two stand-alone graphics editors and two composite, desktop publishing and graphics editors. The stand-alone packages are:

- Artisan (Media Logic Incorporated), \$1,795 gray-scale, \$3,250 color.
- Graphics Builder (Xerox Corporation), price not listed.

Both of these provide standard graphics creation capabilities such as drawing in color with different brushes, diagram making, etc. Differing from most PC graphics packages, these

workstation packages offer image processing functions for scanned images such as image composition, color correction and enhancement.

Desktop publishing packages are more prevalent on the workstation. These packages almost always include some form of graphics generation and manipulation. The following are two examples of desktop publishing packages that include a graphics production system:

- FrameMaker (Frame Technology Corporation), approximately \$2,500.
- Publisher (Arbor Text, Inc.), \$1,995.

The graphics production systems in these composite packages allow the user full-function paint and draw capabilities like one would find in the MacPaint and MacDraw programs on the PCs.

4.3.3 Standard Graphics File Formats

It is obvious that with many different scanners, image processors, graphics packages, and desktop publishers on the market, some standard in image file formats had to evolve. There are several formats that are common, including PCX, PIC, IMG, TIFF, and PostScript. TIFF and PostScript have emerged as solid standards, the leader being PostScript. It would be useful to review both of these formats individually.

4.3.3.1 Tagged Image File Format (TIFF)

Tagged Image File Format has become a standard for scanned image formats. The standard was originally developed on the Apple Macintosh and is very popular because it provides a format for porting image files between PC and Mac graphics packages.

TIFF uses tags to identify graphic elements. The simplest component of the TIFF format is a bitmap of an image. The TIFF format also stores information about the gray-scale or color of each dot in the image. The TIFF format can still be used, however, if the software you use does not provide gray-scale or color information. In addition, TIFF allows for the simultaneous storage of multiple resolution bitmaps of the same image in the same file. Thus, software can choose between the multiple resolutions depending upon the display used.

According to the October 1987 issue of PC Magazine, the big disadvantage of bitmap files is that most software arranges pixels into halftone dots, freezing the print resolution of the image. But if the software captures gray-scale information, TIFF stores it and makes it available to the software for manipulation.

4.3.3.2 PostScript Format (PSF)

PostScript Format and its successor Encapsulated PostScript (EPS) have become the leading standards for image formats across all hardware and software platforms. PostScript became the standard for text in conjunction with the word processing and laser printing fields. Its successor EPS stores images with a kind of vector format that is based on PostScript. EPS is used in graphics as well as text and is a fundamental part of the leading graphics packages on the market today.

4.4 Computer Aided Design (CAD)

For a number of years, the biggest use of PC and mini-computer graphics has been in the area of Computer Aided Design (CAD). CAD packages allow designers to interactively design whole objects or parts of objects by specifying the objects dimensions. CAD systems allow the designer to view the object under design from different viewpoints and to incorporate modifications into the design very easily. CAD systems allow designs to be produced more quickly, and they also produce renderings that are more realistic than pencil drafts. Electronically stored CAD designs can also be more easily copied and modified than pencil designs.

4.4.1 CAD Goals

The fundamental purpose of CAD packages is to capture a schematic or design from the imagination of the designer. Most of the CAD tools have focused on interacting with the designer to provide him/her with ease of input, retrieval, and manipulation of his/her designs. Many CAD software packages have elaborate facilities which allow design and display of 3-D components or schematics with easy-to-use interfaces and plenty of on-line help.

CAD software allows part libraries to be formed which can be incorporated as a portion of a larger design or reused in other designs. Design elements can be assigned names and other information that give meaning to them. Relationships between elements can be specified as well.

CAD systems are used in a wide variety of fields:

- Electrical engineering
- Automobile and aircraft design
- Architectural and building design
- Communications network design
- Water and electricity supply systems design

CAD systems allow designers to see objects on a computer screen before they are constructed. CAD systems often work in parallel with simulation software to allow the designer to exercise objects before they are constructed. In fact, advanced IC design CAD systems allow the schematic that is produced to be tested with simulation software which allows verification of the correctness and functionality of the design. Many architectural CAD systems allow the user to take a "tour" via computer through the structure being designed.

4.4.2 CAD Packages for the PC

PC CAD systems are almost as popular as word processing systems, and as a result there are numerous CAD systems available to meet every budget and need. PC CAD packages are available for as little as \$2.00 and costing as much as thousands of dollars. There are two leaders in the PC world in CAD software, namely, AutoCAD and VersaCAD. AutoCAD has the advantage of being a pioneer and a mainstay in the PC CAD world, a tool against which other systems are measured. VersaCAD, while not as widely used as AutoCAD, has earned the reputation of being a more user-friendly system. Both systems essentially have the same features or functionality. VersaCAD, however, leads in allowing new users to learn and start working quickly, while AutoCAD commands are said to be cryptic and difficult

to learn. Each of these packages can read and write IGES files as well as AutoCAD DXF files, and both can be purchased at most PC software retailers. See Section 4.4.4 for a discussion of the various file formats.

4.4.3 CAD Packages for the Workstation

As in the PC arena, there are numerous CAD packages for the workstations. A quick look at the Sun Microsystems Catalyst Catalog reveals many different CAD systems for just the Sun hardware platform alone. Numerous other packages exist for the other workstation hardware platforms. The workstation market differs from the PC market in the number of general purpose CAD packages available. Workstation CAD packages are usually targeted toward a very specific CAD function (IC design is a popular one). CAD packages for the workstation are typically much more expensive than their PC counterparts. The following is a list of several workstation CAD packages:

- Allegro PCB Design System (Valid Logic Systems Inc.)
- Analog Design and Analysis Environment (Daisy/Cadnetix Inc.)
- AutoPCM (Cadisys Corporation)
- CAECO Designer (Silicon Compiler Systems Corporation)
- Dash Schematic Designer (Data I/O Corporation)

4.4.4 Standard CAD File Formats

As in the word processing arena, almost every CAD system uses a proprietary file format. If there are a thousand CAD systems, there are a thousand different formats in which CAD designs are electronically stored. In order for one CAD system to share designs with another CAD system, a CAD electronic file format standard had to be built.

4.4.4.1 EDIF/IGES

An attempt at defining a standard CAD file format was made in the mid 80's. This standard was formed or supported by several large companies including, Motorola, National Semiconductor, Mentor Graphics, Tektronix, Texas Instruments, AT&T, and Hewlett-Packard. The standard was called the Electronic Design Interchange Format (EDIF), but it was such a loose specification, that it took many different evolution paths. EDIF ended up being not much more than a suggested way of describing CAD files.

A follow-on standard which has succeeded in becoming a standard is the Initial Graphics Exchange Specification (IGES) which was defined by the U.S. Department of Commerce. This standard established the information structures to be used for the digital representation and communication of product definition data (CAD designs). The file format that was decided upon treats the design definition as a file of entities, each entity being represented in an application-independent format. Each specific CAD system can then map between the IGES files and its own native representation.

To do this, the CAD designs are described in terms of geometric and non-geometric information, with non-geometric information being divided into annotation, definition, and organization. The geometric data consists of a definition of the design in terms of points, curves, and surfaces. The annotation, on the other hand, consists of those elements which are used to clarify or enhance the geometry, including dimension, drafting notation, and text. The defini-

tion category provides the ability to define specific properties or characteristics of individual or collections of data entities. The structure category identifies groupings of elements from geometric, annotation, or property data which are to be evaluated and manipulated as single items.

One of the entities that is defined as part of the IGES standard is of particular interest to a later discussion in this paper. The IGES standard contains a provision which is suitable for saving the meaning (or information that leads to meaning) of CAD images. The Property Entity allows non-geometric numeric or textual information to be related to any entity. Any entity occurrence may reference one or more property entity occurrences as required. A CAD file stored in the IGES format, then, can contain information that would lead to understanding of the entities of the design (i.e., bubbles, squares, and edges).

Another entity of interest is the Implementor Defined Entity. This specification allows implementors to include entities in their IGES files that are not defined specifically in the IGES standard. It is, in effect, a generic entity that can be used by CAD programs to define non-standard things. This type of entity could be used as a hook, for example, to hang interesting information that might be needed to interpret designs from IGES file formats.

4.4.4.2 DXF

Another standard that has come about as a result of market competition is the DXF format that is used to save designs from the PC's AutoCAD package. This file format is also used by other competing CAD packages due to AutoCAD's dominance and early presence in the PC CAD arena.

5.0 Image Recognition

This paper has discussed how images (i.e., engineering diagrams) can be scanned into an image file and how that image file can then be manipulated or enhanced through graphics packages, CAD systems, and image processing. Until now, little mention has been made of how to interpret what the scanned and processed image contains that is of meaning. The focus of image recognition is separating the crude bitmap image into its component entities and determining their identity. This may involve identification of shapes, objects, or character text.

There are several areas in which image recognition has been developed into commercially viable products. Two of the applications that have received much attention are Optical Character Recognition (OCR) and Computer Aided Design (CAD). This section will provide an overview of the capabilities in these two areas and a survey of what is currently available in the market place.

The field of image recognition is still an advancing research topic, and there are many unanswered questions and unsolved problems. This chapter will address existing technologies and commercially available applications. The following chapter will address current research in the field of image recognition.

5.1 Optical Character Recognition (OCR)

Optical character recognition is the ability of a computer (whether in hardware or software) to take a scanned image of a page of text and interpret the characters that are in the scanned image to produce an ASCII (or EBCDIC) output file that is the same as the text on the original page. The ASCII file can then be edited by word processors as a document, or the text file can be used by other programs to interpret the meaning of the characters or words.

5.1.1 Origins

Optical character recognition grew in conjunction with the demand of the word processing industry to have an automated facility to input documents from paper to a digitally stored format. With massive amounts of documents to store and process via computer, it was justifiable to find an automatic alternative to hiring typists to enter the text. Since OCR is so intricately associated with the necessary process of digitizing the text image, 25% of all scanners that are currently produced have some OCR capability. Often the bundled software is tailored to the associated hardware. There are other OCR software packages that do not require specific scanner hardware that can read from standard image file formats.

5.1.2 How it Works

Whether the recognizing of characters is done in the scanner hardware, on an expansion card in the computer, or in the software alone, OCR can be divided into three categories based on the algorithms used to recognize individual characters:

- Matrix matching
- Feature extraction
- Hybrid methods

OCR is very important in the solution of automatically "scanning" logic diagrams. Logic diagrams similar to the one in Figure 1-1 contain a relatively high percentage of text, so these methods will be investigated individually.

5.1.3 Matrix Matching

First, most of the low-end OCR software packages use matrix matching, sometimes called font recognition. The OCR software contains a fixed pixel matrix for each character which can be recognized. The OCR starts at the top of the image and isolates a group of pixels using component separation. Component separation is performed using the assumption that each character will be composed of a tightly packed pattern of black pixels with a rectangular area no larger than the template characters. The isolated patterns are often called MRAs, Minimum Regions of Analysis. The isolated pattern of pixels is then matched against each of the known character matrices until a match is found. If the character does not match any of the known characters, the OCR software may choose to ignore the character or ask the user for interpretation.

Matrix matching OCR software has proven useful when recognizing monospaced characters from a restricted number of character fonts and type sizes. Matrix matching software is often capable of "learning" a new character font. The new font may be appended to the existing character matrices or may replace existing character matrices. The time to recognize each character increases and the accuracy of correct character recognition decreases as the number of known character matrices is expanded.

Matrix matching is not robust enough to recognize a new font without having been "trained" on that font. The matrices of each character in each new font must become a member of the set of matrices with which the OCR software compares. Furthermore, if this algorithm is expected to recognize characters with tilted orientations, it must be trained on all orientations of all characters of all fonts. This is obviously prohibitive, and matrix matching OCRs usually cannot recognize tilted text.

5.1.4 Feature Extraction

The next category of recognition methods, feature extraction, is usually used by more expensive OCR software. Feature extraction, sometimes called feature recognition, is based on the principal that every character has a set of distinct features which can be used to identify it. As a document is scanned, an MRA is identified and the features of a potential character within the MRA are determined. These features are compared against the features of known characters until the character is either identified or rejected.

Feature extraction is a much more powerful recognition technology than matrix matching because it is more immune to the effects of image noise, character point size, character font, and character spacing. Complex and decorative fonts may not be recognizable by feature extraction due to the extraneous characteristics for each character. Feature extraction software is more complex to develop than matrix matching software and often is enhanced by the use of context information to improve the accuracy of the recognition process.

Feature extraction software would identify the MRA "q" by determining the region's characteristics. The MRA "q" consists of a circle attached to a vertical line with the circle being located at the top of the vertical line. The letters "q" and "d," and the number "9" consist of a circle on the left of a vertical line. Due to the fact that the vertical line is straight, the number

"9" is eliminated as a possibility. The letter "d" is eliminated as a possibility due to the fact that the circle portion of the character occurs at the bottom of the vertical line instead of at the top.

5.1.5 Hybrid Methods

Alone, both matrix matching and feature extraction OCR produce several errors per page. Finally, a hybrid method of recognition which combines parts of both matrix matching and feature extraction can be used which results in fewer undetected errors. To recognize a character, the hybrid OCR algorithm would check first with a matrix to find an obvious match, then it would look for features that are characteristic of that character to solve any dispute. As a further check, a hybrid OCR algorithm takes the words it finds and looks for them in a dictionary to make sure they exist. Misinterpreted characters are found by the misspelling of the word to which they belong. For example, "The dog sut down" would be found to be "The dog sat down." Needless to say, the hybrid algorithms, though more accurate, can be processor, storage, and time intensive. All algorithms require a human proofreader to catch all of the errors.

5.1.6 Commercially Available OCR

There are an increasing number of OCR packages available. Some are stand-alone software packages, while others come standard as part of an optical scanner.

5.1.6.1 OCR for the PC

According to a DataPro Review in December 1988, the Calera Character Recognition Systems' TrueScan (\$2,495) is a good buy because of its low error rate, even on dot matrix quality documents. The software also recognizes lines, tabs and columns. The software outputs the text in a variety of popular word processing formats.

An October 1989 DataPro Review praised the Kurzweil Character Recognition System (Xerox Imaging Systems) products which range from \$17,950 on the high end to \$2,995 on the low end. This company has a long history of good products in this industry, and the systems run on standard microcomputers. The degree of accuracy can be adjusted to the job by the user. These systems, however, do not support UNIX-based workstations.

5.1.6.2 OCR for the Workstation

Neither the most recent MassComp (Concurrent Computer Corporation) Solutions Catalog nor the latest Sun Microsystems' Catalyst Catalog mention OCR software for the workstation. This is understandable, since it would be trivial to scan in a document on a PC and then upload the document text file to a workstation word processor. Perhaps the price of workstations still prohibits their use as extensive word processing tools.

5.2 CAD Overlaying

In addition to OCR, the need for comprehensive CAD tools has been a driving force in the application of image recognition techniques in commercial CAD software. Though the principle use of CAD software to capture schematics from designers needs no image recognition, the need to capture designs that are already existing on paper into electronic formats has driven the industry to develop some software that requires image recognition.

Many CAD systems provide the ability to bring paper printed designs in through a technique called CAD overlay. The technique is not automatic at all; in fact, it is only a tool to aid in accelerating human reentering of the design. The advantage of this method of reentry is that a less skilled person could perform the job without having to understand the design.

5.2.1 How it Works

To bring a preexisting design into VersaCAD using the overlay technique, for example, one would first scan the drawing into an image file using an optical scanner and its related software. Then one would bring the drawing onto the VersaCAD screen as a background image. Next, the user would choose system-defined (or user-defined) objects such as squares, lines, and bubbles to overlay on corresponding objects in the image. One essentially recreates the design in the VersaCAD system by piecing together (overlying) objects on the screen. The scanned image acts only as a back-drop to guide a human in the overlay (recreation) process. The result is that the design is transferred from paper to an electronic file format much more quickly than if it were redesigned without using overlays.

5.2.2 Commercially Available CAD Overlay

Both the AutoCAD and VersaCAD systems have software available to them that allow overlay design entry. AutoCAD has a supplemental package called CADOverlay, and VersaCAD has this supplemental software included in it.

5.3 Raster-to-Vector Conversion

The demand for a more automatic way of entering old designs into CAD systems and to store them electronically has brought about the need for computers to be able to "look" at a scanned image and distinguish the entities that make up that design which might be bubbles, squares, lines, text descriptions, etc. Capturing the graphics entities which make up the drawing is done by a process called raster-to-vector conversion. It must be made clear that even though raster-to-vector conversion captures the graphics entities, it still has not captured the meaning of those entities. This means that the NASA problem will require specialized software that is not commercially available to interpret meaning, even if a logic design could be scanned into an electronic format automatically with off-the-shelf software packages.

A scanned image is in a form called raster format, meaning it is just a bunch of bits that represent pixels or dots in the picture. CAD systems, on the other hand, do not store graphics information in raster format, but they use vector information to describe graphics objects. For example, a vector stores a line as a starting point and a length. The other graphics objects, such as a square, have similar vector definitions.

For a computer, the task of interpreting the scanned image into vector format is not trivial. With the advances in image recognition, edge identification and other such algorithms have been developed to the point that software can convert from raster to vector for CAD input. Depending upon the quality of the original, the conversion will have more or less errors. Almost all conversion will have errors that need to be touched up by humans. Nevertheless, this conversion and correction is much faster than manual entry.

5.3.1 Raster-to-Vector Conversion Packages for the PC

The only raster-to-vector software available on the PC is tightly associated with CAD packages. The following are three examples of available PC packages:

- CAD/camera (Autodesk Corporation) \$3,000
- CADmate (Microteck Corporation) \$995
- ScanPro (American Small Business Computer Corp.) \$495

CAD/camera is the pioneer in the field of commercial raster-to-vector software, but without doubt, ScanPro is the best of the three. ScanPro is faster and more reliable. CAD/camera is directly associated with AutoCAD. ScanPro can also produce the vectors in AutoCAD's DXF format.

6.0 Image Processing/Image Recognition Research

There are currently numerous research projects involving image processing and image recognition. A quick look at the number of papers published in the IEEE Transactions on Pattern Analysis and Machine Intelligence indicates the interest in image processing and image recognition. The following papers were chosen because of their relevance to the proposed solution to the NASA problem. These papers deal with separation of text and graphics and graphics symbol recognition.

6.1 Research in Graphics/Text Separation

"A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images"
IEEE Transactions on Pattern Analysis and Machine Intelligence

The discussion in this research paper is directly related to the proposed NASA solution for several reasons. The paper outlines a process for separating text strings of various font styles and sizes from graphics images which contain both graphics symbols and text strings. This is ideally suited for the NASA solution because of the diversity in the content of the logic drawings currently developed by the different groups at NASA. The paper also provides experimental results which quantify the effectiveness of the algorithm using a Microtek Image Scanner and an IBM PC-AT. The Microtek scanner provides similar functionality to the scanner which NASA has already interfaced to a MassComp workstation. Much of the research being conducted in the image processing and image recognition fields is performed on custom or very expensive hardware. For example, the diagram reader discussed in the next section utilizes four processors and custom wired-logic hardware. The hardware used in this research is very similar to the hardware which is readily available currently at NASA.

Engineering diagrams and logic drawings usually contain graphical symbols and supporting textual labels or descriptions. Fletcher and Kasturi first attack the problem of interpreting drawings and diagrams by attempting to separate the graphics symbols from the text. Two image files are generated from each graphics image of an engineering diagram or logic drawing. The first image file contains the text strings which were in the original graphics image, and the second file contains only the remaining graphics symbols. This allows OCR systems to operate on an image file containing the text strings, and graphics recognition systems to operate only on the graphics symbols. The algorithm described and evaluated in the research paper focuses on the effectiveness of the graphics/text separation.

The Fletcher and Kasturi algorithm is unique in its ability to locate text regardless of the text's orientation, font style, or font size. The algorithm places several restrictions on the text within a drawing, but these guidelines are less restrictive than previously implemented algorithms. For a comparison, note the restrictions placed on the text strings in the following research paper.

The algorithm for separating the text strings from the graphical image consists of the following steps:

- Connected component generation
- Area/ratio filter
- Collinear component grouping

- Logical grouping of strings into words and phrases
- Text string separation

The method uses eight connected pixels in the image to find connected pieces of a character. It assigns a bounding rectangle to the connected pixels (the character or a small graphics item) by using the minimum and maximum x and y coordinates associated with the group of pixels. The character or graphics item is then effectively blocked off.

When all the groups that are thought to be characters on the page are bounded, a search is made to find "strings" of text by comparing the centroids of the many bounding boxes. If a given box is found to be collinear with other boxes, it is assumed to be a character, and the entire string is extracted from the mixed image and placed in a separate text image file. With a separate file containing just text, an OCR system can be used on the text image file to determine individual characters. The remaining graphics image file can be converted using the raster-to-vector conversion systems discussed in Section 5.3.

This current research only addresses the issues of separating text and graphics and interpreting each into their respective ASCII and vector codes. It does not address the subject of understanding the logic represented by the graphics symbols or the meaning of the text within the drawing. It does provide a robust algorithm for the first step in the automatic scanning and interpretation of the NASA logic drawings.

6.2 Research in Reading Engineering Designs

"An Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition"
IEEE Transactions on Pattern Analysis and Machine Intelligence

At Toshiba, much of the initial Large Scale Integration (LSI) engineering design work is still done on paper by engineers. These initial designs drawn on paper are then input into CAD systems for analysis, modification, and production. The process of inputting the hand-drawn engineering diagrams into the computer based CAD system is a time consuming process. An error correction process often follows the transfer from paper to computer.

Five researchers at the Toshiba Research and Development Center in Kawasaki, Japan, have reduced the time consuming process of inputting an engineering diagram into a computer by implementing a custom logic circuit diagram reader for VLSI-CAD data input. The logic circuit diagram reader automatically scans the engineering diagram, interprets the symbols and text of the diagram, and outputs the information contained in the diagram in a format that is usable by a CAD system.

The diagram reader developed by Toshiba is a circuit diagram reader only, and traditional circuit diagrams are composed of two types of symbols: loop symbols and loop-free symbols. Loop symbols are those circuit diagram symbols which are composed of one or more closed loops. The various gate symbols, such as AND and OR gates, are examples of closed loop symbols. Loop-free symbols are those circuit diagram symbols which are not composed of closed loops. The GROUND and RESISTOR symbols are examples of loop-free symbols. Many of the closed-loop symbols used in circuit diagrams are very similar in design with only subtle differences. The circuit diagram reader developed by Toshiba was designed with great emphasis on the correct interpretation of closed-loop symbols.

Closed-loop symbols are recognized in a two stage process. The first stage involves locating and delimiting the regions which contain closed-loop symbols. The Toshiba researchers call this first stage "symbol segmentation." Minimum regions of analysis (MRA) are determined in much the same way they are determined in OCR software. This is possible because each symbol's shape and size is known, due to the fact that the Toshiba prototype requires the diagrams to be drawn using a template.

In the second stage, the delimited regions are analyzed to determine which closed-loop symbol is contained within each region. The Toshiba researchers call the second stage "symbol identification." The second stage uses template matching and feature extraction, also used in OCR software, to perform symbol identification. A decision tree performs the template matching and feature extraction in parallel on each MRA. The decision tree uses the output of both identification methods to identify the symbol. The decision tree allows the software to utilize the advantages of both identification techniques without incurring the inaccuracy of an individual technique.

Figure 6-1 displays the flow of Toshiba diagram reader.

Decision trees allow new symbols to be easily incorporated into both stages. Decision trees also allow the accuracy of the two stages to be improved as more is learned about each stage and the processes are refined.

The Toshiba prototype developers found that the greatest difficulty in introducing their LSI circuit diagram reader was deciding on the drawing rules. The LSI designers requested that the additional drawing rules imposed by the diagram reader be as few as possible. The following rules were imposed to ensure accurate reading of the diagram within reasonable time limits:

- A template of the allowable logic symbols must be used.
- Eight orientations of each symbol are allowed: symbols rotated 90 degrees and their reflections.
- The lines connecting symbols must be drawn with a ruler.
- The shortest allowable line segment length is 2 mm.
- Character labels must be written in a specific font recognized by the OCR portion of the diagram reader.
- Each character must be no larger than 4 mm x 4 mm.
- Characters may be written in two orientations: left to right horizontally and bottom to top vertically.
- Diagram symbols must be at least 1 mm apart, and line segments must be at least 2 mm apart.

The circuit diagram reader implemented by Toshiba is a combination of custom image processing hardware and software. The image input device is a drum scanner which scans 10 lines/second at a resolution of 10 lines/mm. The preprocessing, component separation, and loop-symbol detection portions of the diagram reader are implemented using custom wired-logic circuits. The other processes are performed on two microprocessors and two minicomputers utilizing pipeline and multiprocessing techniques.

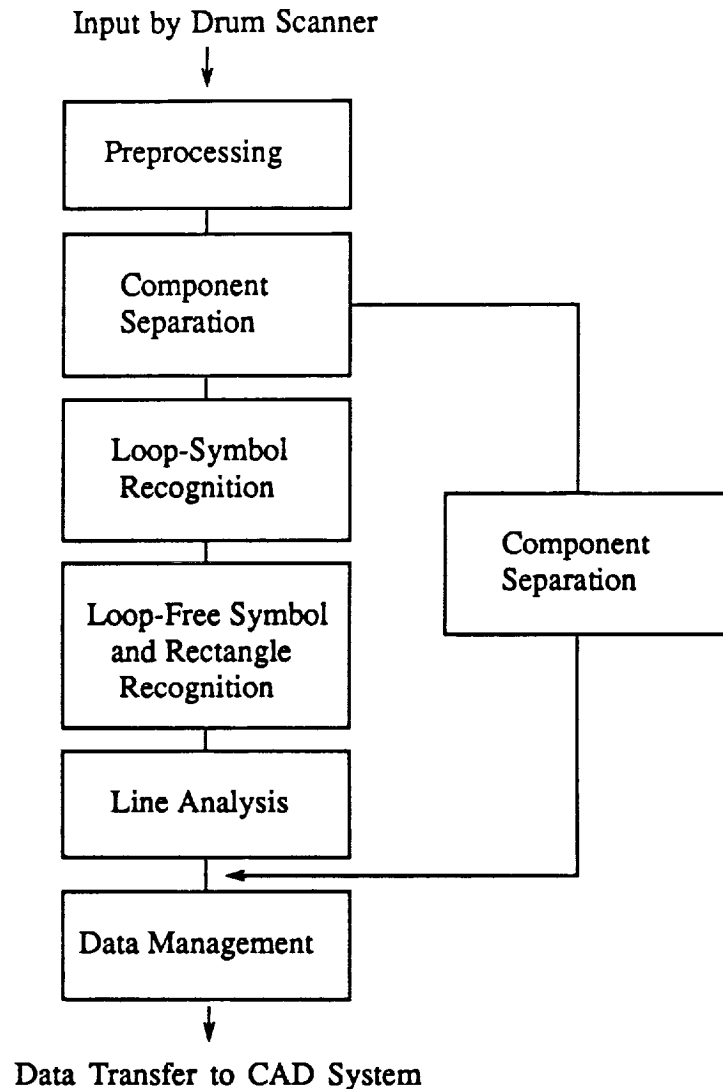


Figure 6-1 Toshiba Diagram Reader Flow

The Toshiba diagram reader has been tested on 851 drawings of various sizes, and 79 gate-array LSI's have been developed from drawings read by the diagram reader. The diagram reader can read an A2-size (420mm x 594mm) diagram within 20 minutes. On average, 29 error corrections or rejection modifications are required for each A2-sized diagram. The process of reading an A2-sized diagram and interactively correcting errors or identifying rejected symbols usually takes about 30 minutes when using the diagram reader. Interactively entering a comparable A2-sized drawing without using the diagram reader usually takes about one hour.

A series of ten A1-sized (594mm x 841mm) and A2-sized diagrams were read using the diagram reader. The diagrams were prepared using the established drawing rules and a char-

acter template to improve OCR. Of 2,488 possible symbols, 96.5 percent of the symbols were read correctly, and 2.8 percent of the symbols were unrecognizable by the diagram reader. Only .7 percent of the symbols were incorrectly identified.

The Toshiba reader is an example of a successful image processing application. The Toshiba diagram reader accurately reads logic-circuit diagrams without imposing a large set of unreasonable drawing rules. The diagram reader suffers from the OCR character template, which is not required but recommended, and the amount of custom hardware used to implement the diagram reader. The NASA solution could benefit from the Toshiba concept of melting several techniques to produce a more powerful product.

7.0 Possible Directions Toward A Solution

In light of the concluded survey of commercial optical scanning, image processing, and image recognition systems, several scenarios are possible that would improve the process of converting logic drawings to software. The proposed solution should meet several general guidelines:

- The solution should support the various groups at NASA
- The solution should not require extensive custom hardware
- The solution should address the ultimate goal of automatic generation of exhaustive test algorithms

The preferred scenario would be to scan in the logic drawing using a scanner which is connected to either a NASA workstation or IBM PC. The imported image file would then be processed to improve its quality. Image recognition software would then process the enhanced image file producing a logic description file. This description file would contain all of the logic contained in the original logic drawing and any associated text. The description file would then be converted into a Comp Builder (Computation Development Environment) file which would be batch compiled. The final result would be an installed computation ("comp") and a file describing an exhaustive method for testing the "comp." Such a system does not commercially exist, and research is only now beginning to show progress toward such a solution.

Several intermediary solutions are also possible. The automation path that might be taken is to scan in the logic design using a scanner and the necessary image processing associated with it. Then, convert the raster image to a vector image that can be used in a CAD system. Hopefully, this could be done completely by a system like ScanPro without the use of any overlay package like CADOverlay or the overlay capabilities of VersaCAD. This solution would definitely require human interaction. Existing import facilities are limited in their ability to automatically interpret image files. The CAD design could then be saved as an IGES or DXF file (the DXF file contains more meaning associated with each entity, but the IGES file format is more of an industry standard). At this point, the logic design is described in electronic description files. Custom software would then interpret the description files and produce a Comp Builder file. The remaining steps in the solution would follow the first solution.

Another intermediary solution would be to use computer based software to build the original logic drawings. In this way, a description file would be produced from the user's input instead of from an image file. NASA and its various groups currently use a variety of software based on both PCs and workstations to generate the logic drawings. The solution software must operate on the various group's existing hardware and gain wide-spread acceptance. This solution has been investigated by NASA, and a prototype based on the G2 Real-time AI package has been developed.

8.0 Referenced Vendors

Apple Computer, Inc. 20525 Mariani Ave. Cupertino, CA 95014 (408) 996-1010

Calera Recognition Systems, Inc. 2500 Augustine Drive Santa Clara, CA 95054 (408) 986-8006

Chinon America, Inc. 660 Maple Ave. Torrence, CA 90503 (213) 533-0274

Concurrent Computer Corporation 4828 Loop Central Drive Suite 550 Houston, TX 77081 (713)

Hewlett-Packard Co., Business Computing Systems 19091 Pruneridge Ave. Cupertino, CA 95014 (800) 752-0900

Howtek Inc. 21 Park Avenue Hudson, NH 03051 (603) 882-5200

Logitech, Inc. 6505 Kaiser Drive Fremont, CA 94555 (415) 795-8500

Media Cybernetics (Dr. Halo) 8484 Georgia Avenue Silver Spring, MA 20901 (301) 495-3305

Microsoft Corporation 16011 NE 36th Way Box 97017 Redmond, WA 98073-9717

Microteck Lab, Inc. 680 Knox Street Torrence, CA 90502 (213) 515-3993

Mitsubishi Electronics America Inc. Computer Peripherals Division 991 Knox Street Torrance, CA 900502 (213) 217-5732

Sun Microsystems, Inc. 2550 Garcia Ave Mountain View, CA 94043 (415) 960-1300

Xerox Imaging Systems Kurzweil Division 185 Albany St. Cambridge, MA 02139 (617) 864-4700

For AutoCAD, VersaCAD, CADOverlay, CAD/camera, and ScanPro: See local retailers for pricing and address information.

9.0 Bibliography

- "All About Scanners: Technology Overview," DataPro, May 1989.
- "Calera Character Recognition Systems," DataPro, May 1989.
- Catalyst Fall 1989 Sparc Edition, Sun Microsystems, Inc.
- Fletcher, LLoyd ..., "A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 6, Nov. 1988.
- Hearn, Donald, Computer Graphics, Prentice Hall, Inc. Englewood Cliffs, NJ, 1986, pp. 2-26.
- "Hewlett-Packard ScanJet Plus," DataPro, May 1989.
- "Kurzweil Character Recognition Systems (Xerox Imaging Systems)," DataPro, Oct. 1988.
- "Microteck Scanners," DataPro, April 1989.
- Okazaki, Akio ..., "An Automatic Circuit Diagram Reader with Loop- Structure-Based Symbol Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 10, Number 3, May 1988.
- "Scanners Build A Better Image," PC Magazine, Volume 8, Number 6, March 29, 1989.
- Smith, Brad, Initial Graphics Exchange Specification (IGE), Version 3.0, U.S. Department of Commerce, National Bureau of Standards, National Engineering Laboratory, Center for Manufacturing Engineering, Automated Production Technology Division, Gaithersburg, MD, April 1986.
- Solutions Sprint 1989, Concurrent Computer Corporation.

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

FLIGHT CERTIFIED COMPILER

NASA Grant No. NAG 9-339
SwRI Project No. 05-2768

Prepared by:
Steven W. Dellenback, Ph.D.
Jeremiah M. Ratner

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

June 12, 1990

Approved:

Susan B. Cremone

for Melvin A. Schrader, Director
Data Systems Science and
Technology Department

Table of Contents

1.0 Introduction	1
2.0 Flight Certified Software	2
2.1 NASA Flight Certified Software	2
2.2 Flight Certified Software Guidelines	2
2.3 Language Alternatives	2
3.0 Language Definition	4
3.1 Context-Free Grammars.....	4
3.2 Chomsky Normal Form	6
3.3 Griebach Normal Form	6
3.4 LR Grammars.....	7
3.5 Conclusion	7
4.0 Language Implementation.....	8
4.1 Language Specification.....	8
4.2 Implementation Alternatives.....	8
4.2.1 Machine Code Generation	8
4.2.1.1 Advantages.....	9
4.2.1.1.1 Implementation Simplicity	9
4.2.1.1.2 Performance	9
4.2.1.2 Disadvantages	9
4.2.1.2.1 Language Restrictions.....	9
4.2.1.2.2 Limited Value Added.....	10
4.2.1.2.3 Error Recovery	10
4.2.2 Table Driven Generation.....	10
4.2.2.1 Advantages.....	10
4.2.2.1.1 Error Recovery.....	10
4.2.2.1.2 Semantic Analysis.....	10
4.2.2.1.3 Interactive Debugging.....	11
4.2.2.1.4 Portability.....	11
4.2.2.2 Disadvantages	11
4.2.2.2.1 Performance	11
4.3 Implementation Considerations	11
4.4 Conclusions.....	12
5.0 Verification	13
5.1 Grammar Specification Verification.....	13
5.2 Language Implementation Verification	13

6.0 Conclusions	14
7.0 Bibliography	15

1.0 Introduction

Development of flight software at NASA is currently a tedious and time consuming process. One of the more difficult (and nebulous) phases of the development cycle is the certification phase. During this phase, a decision is made as to whether the software can be used during flight operations (i.e., the software is deemed "flight certified").

The typical operating scenario at a console during flight operations at NASA - Johnson Space Center (JSC) involves a flight controller interpreting the real-time data screens downloaded from the vehicle. With recent advances in graphics technology, an effort is underway to provide the flight controller with software which aids in the understanding/interpretation of the data.

A major restriction in the current operations scenario at NASA-JSC is that programs must be (typically) developed by trained programmers. This approach has limitations because the flight controllers must convey to the programmer what actions the software must perform. Many times programmers do not correctly interpret the requirements given to them and develop software which is not exactly what the requestor had envisioned. This limitation is being addressed through the development of simplified programming languages/interfaces (e.g., User Interface Language (UIL), Computation Development Environment (Comp Builder or CODE), etc.) which will allow a nonprogrammer to develop application programs in a controlled environment. While these "user friendly" languages will provide a simplified programming environment, they still will have the restriction that all coding must be performed before flight operations unless some automated certification tool is developed.

This paper discusses techniques which, if implemented, could allow a programming language (if the language is carefully designed) to generate programs during flight operations. The goal of this paper is to discuss the concepts and techniques that would allow the implementation of a language which would assure that a user's program is reproducible and safe (defined as behaving as implemented). The tool described in this paper is termed "flight certified compiler," a term which implies that the tool would develop flight certified software from a noncertified software source code. Criteria for what would constitute a "flight certified compiler" are discussed in Section 2.0 of this document.

The term compiler is used throughout this paper to discuss a software tool which will convert a source language into a target language (not necessarily machine language). The word compiler and translator can be used interchangeably throughout this document.

The flight certified compiler concepts discussed within this paper can be applied to a multitude of target environments. While the original impetus for this paper was based on MCC workstations, the compiler concepts discussed are generic and are not based on a particular operating system or hardware platform. As a result, the concepts presented in this paper can apply to mini-computers as well as mainframes. A worthwhile flight certified compiler should actually operate in all computing environments at NASA so that users are not restricted to a particular operational environment.

2.0 Flight Certified Software

Any software which is run in the Missions Operations Computer (MOC) during flight operations must be "flight certified." Flight certified software, in its traditional sense, is defined as software which has been reviewed by appropriate NASA personnel and, upon execution, will generate correct results and provide reproducible results (given similar input).

2.1 NASA Flight Certified Software

The NASA Flight Control Operations Handbook does not elaborate on how a piece of software becomes flight certified; rather, the handbook leaves the process to the discretion of each Division of NASA.

As a result, for the purposes of this paper and research study, the following section will establish guidelines for what constitutes a flight certified compiler.

2.2 Flight Certified Software Guidelines

The following criteria are recommended as the necessary and sufficient qualifications to term a compiler to be flight certified:

- The target language generated must be reproducible (i.e., the same input WILL ALWAYS generate the same output).
- The target language, when executed, will cause no abnormal events to occur within the host computer or to remote computers attached via a Local Area Network (LAN).
- No combination of source data structures/statements shall cause a program to generate incorrect results (i.e., the program will ALWAYS compute as directed by the user).

In order to achieve each of the above criteria, certain operations must be performed to assure that the grammar behaves as desired. The remainder of this paper discusses techniques and implementation strategies which can be applied in order to develop the desired flight certified compiler.

2.3 Language Alternatives

Over the last five years, several universal programming languages have been defined which have potential uses within the NASA-JSC MCC environment. The most notable is the User Interface Language (UIL) which is an object oriented approach to software development. The reader is referred to the UIL Language Specification for a more detailed discussion.

Another language specification under development by NASA-JSC is the Mission Operations Applications Language (MOAL) which is a procedural oriented language to be used in the development of simple computational algorithms.

While the goal of this paper is not to establish a particular language for the flight certified compiler, the prospective language should contain the following classes of constructs:

- Named, strongly typed variables/constants
- Assignment operators
- Loop constructs (pre- and post-conditional loops)
- Conditional constructs (single, and multi-case selection)
- Modularization constructs (e.g., functions)
- Data acquisition constructs (provide a mechanism for programs to efficiently and reliably acquire LAN data)
- Input/output primitives (for retrieval and display of data)

The reader will note that most of the above constructs exist in many higher order languages currently implemented. The goal in developing the flight certified compiler is to provide a target language which would allow users (nonprogrammers) to develop their own software in a highly structured environment. As a result, the remainder of this paper will focus not on the specific constructs of the language; rather, the paper will focus on what techniques need to be utilized to implement a language and allow the resulting tool to be flight certified.

3.0 Language Definition

The following section will provide the ground work for the establishment of a flight certified compiler. The following sections discuss various theoretical concepts associated with formal language theory. The concepts must be understood and applied to a flight certified language specification before a flight certified compiler can be developed.

The reader must understand the program generation process in order to be convinced that a flight certified compiler is an achievable goal. Simply, the compilation (or interpretive) approach to generating an executable program can be broken into the following steps:

- Scan the input string provided by the user (i.e., the program) into tokens. Tokens are the smallest recognizable element of the grammar being implemented; typically tokens are operators or identifiers which are either organizationally or white space separated.
- The tokens scanned are passed through a parser to determine if the tokens represent a sentence in the target grammar. This is the syntax checking portion of the compilation process. The parser bases its syntactic rules on the way in which the grammar is specified (i.e., all the syntax rules are inherent from the grammar definition).
- Once the string of symbols is verified to be syntactically correct, target code is generated for the provided input language.
- After target code is generated, binding of external modules and variables is performed so that a complete and "executable" module is generated. This completed module may be in the specific form of absolute machine language or could be as general as a table driven environment which requires an interpreter to assist during execution time.

The first step in implementing a flight certified compiler is having the correct definition for the language to be implemented. The language to be implemented must be unambiguous (which will assure the compiler of a single parse tree). The specification of a language syntax is not an obvious and trivial action to perform. It is possible to specify a language which does not provide unique parse trees as well as creating a syntax which never parses (i.e., the parsing process will never complete). Both of these situations are not desirable within the framework of a flight certified compiler.

The following sections discuss techniques which can be utilized to demonstrate that the target language is indeed unambiguous. The discussion begins with brief descriptions of formal notations available to represent the language to be implemented within the flight certified compiler.

3.1 Context-Free Grammars

A context-free grammar (CFG) is notation for specifying the syntax of a language. The hierarchical structure of the language is described in the CFG by a set of terminals, basic symbols from which the strings of the language are formed, and a set of non-terminals, or variables

representing sequences of terminals. Formally, a CFG is denoted with a four-tuple (V, T, P, S) where:

- V is a finite set of nonterminals (variables),
- T is a finite set of terminals (T is disjoint from V),
- P is the finite set of production rules. All productions have a nonterminal on the left hand side with a combination of terminals and nonterminals on the right hand side, and
- S is the start symbol and is a member of the set of non-terminals.

CFGs are characterized by always having exactly one nonterminal on the left hand side of a production. Consider the following example of a CFG:

```
<sentence>::=<noun> <verb>
<noun>::=boy | girl
<verb>::=walks | talks
```

The above grammar would allow the generation of the following strings:

```
boy walks
boy talks
girl walks
girl talks
```

The CFG notation is important to the flight certified compiler because using a CFG, it is possible to programmatically determine if a given string of input symbols (i.e., tokens generated from the program under review) constitutes a language within the definition of the CFG.

It is possible to programmatically determine if a set of input symbols represents an allowable string by the use of a derivation tree. A derivation is rooted at the start symbol and represents the hierarchical structure of the production rules - each leaf represents a terminal. The tree is generated by matching input symbols to the left side of a production and creating a higher level node representing the right side. This process stops when the start symbol is generated.

Depending on the basic structure of the desired language, it is possible to develop a CFG which has productions called "epsilon productions." Epsilon productions are productions which, when applied, do not reduce the input symbol set (i.e., we generate another layer in the derivation tree but do not reduce the input set). Epsilon moves occur naturally in many CFGs, but their inclusion does not assure us of always reducing the input set.

During application of the grammar against a string of input symbols, it is possible within the definition of a CFG to have multiple derivations. If the derivation tree is applied starting at the left end of the input symbols, a left-most derivation tree is generated. Similarly, if the derivation tree is generated by examining the right-most of the input symbols first, a right-most derivation tree is generated. If the two derivations are not equivalent, we have an ambiguous grammar, that is, two paths representing a single string. Since it is not possible to determine programmatically which path was intended (i.e., by the programmer), an ambiguous grammar does not allow for certifiable software translation.

As stated previously, in order to develop a flight certified compiler, we must have grammar that is unambiguous. CFGs have the property that they will allow for ambiguities within the grammar specification. The following section discusses another grammar form which provides us more of the basic utility required to implement a flight certified compiler.

3.2 Chomsky Normal Form

Chomsky Normal Form (CNF) is any CFG without epsilon moves where the productions always have only either nonterminals on the right hand side or only a single terminal. Algorithms exist which will allow the conversion of a CFG to CNF. The motivation to convert from a CFG to a CNF is yet another notational form, Griebach Normal Form, which requires that we have a CFG with no epsilon moves. Epsilon moves are generated when a production is applied but the input set is not reduced by the new production (with equal number of terminals and nonterminals).

An example grammar in CNF form would be:

$$S :: = BA \mid a$$
$$B :: = BB \mid b$$
$$A :: = a$$

with the set of nonterminals being $\{S, A, B\}$, the set of terminals being $\{a, b\}$, and the start symbol being S .

3.3 Griebach Normal Form

Griebach Normal Form (GNF) is any grammar which has no epsilon moves, and all productions start with a terminal symbol and are followed by the number of (possibly zero) nonterminal symbols. The advantage to having a grammar in GNF form is that the input set reduces with every application of a production which will assure that the parsing process will terminate.

An example grammar in GNF form would be:

$$S :: = aAB$$
$$A :: = bB$$
$$B :: = c$$

with the set of nonterminals being {S, A, B}, the set of terminals being {a, b, c}, and the start symbol S.

Grammars in GNF form can be represented by deterministic and nondeterministic pushdown automata. An automaton is a finite state machine which can be used to represent a formal grammar definition. Automata are useful in this discussion because a variety of theorems and collaries exist which deal with the properties associated with automata and prove correct the process by which automata recognizes strings from a GNF grammar and reject all others.

3.4 LR Grammars

The class of grammars known as LR grammars is a restricted notation of a CFG. LR grammar is defined as a CFG with the annotation that the start symbol does not appear on the right hand side of any production and that a set of nonterminals in each production can be uniquely generated from all productions of the grammar.

The major feature of an LR grammar to this paper is that an LR grammar is always unambiguous. Algorithms exist which can convert a deterministic push down to an LR grammar (refer to the bibliography for literature references).

3.5 Conclusion

Given a prospective grammar to be used for flight operations, the language can be analyzed and manipulated in order to structure the grammar in the forms previously discussed. Once the grammar is in one of the forms, certain assertions can be made about the grammar, for example, that the grammar will absolutely produce reliable and reproducible results.

The following chapter of this document will discuss implementation techniques which if applied with these grammar constructs, will provide an environment in which flight certified code could be generated.

4.0 Language Implementation

The implementation of the flight certified compiler can be performed in a variety of computing environments. The certification required of the flight certified compiler falls into two major areas:

- **Language Specification:** The language should be specified in a BNF notation, and the notation should be carefully reviewed before the compiler implementation phase begins. The language should be reviewed for functional completeness by the user community. After review by the user community, the language should be scrutinized to assure that all ambiguities and useless productions are eliminated from the language definition.
- **Compiler Implementation:** The compiler which implements the flight certified language should be a flexible tool implemented to allow the language to be entered in a tabular fashion (i.e., the language constructs are specified in a BNF style so that language modifications are easily accomplished). The design would be to accept the language specification in a BNF form and apply the productions supplied.

Several different implementation techniques exist. This paper will present two different implementation techniques which would be applicable to the compiler. The following sections discuss implementation requirements and suggested strategies.

4.1 Language Specification

The language to be implemented by the flight certified compiler can be a dynamic language; that is, the flight certified compiler should be developed to accept the flight certified language in BNF form. By implementing the flight certified compiler in a table driven fashion, the language can be modified without having to modify the software which implemented the compiler. This implementation strategy will provide a flexible environment which can be changed as needs dictate without forcing the complete recertification of the compiler.

4.2 Implementation Alternatives

Several approaches to implementing the flight certified compiler are available. The language could be executed either in a traditional "compiled" mode (where the source program is translated into machine instructions) or the language could be executed in a "translated mode" (where each statement is interpreted at run-time). Each of the alternatives has advantages and disadvantages which are discussed in the following sections. If funding allows, there would be merit in prototyping both approaches and contrasting them in order to evaluate which provides the better solution.

4.2.1 Machine Code Generation

The first alternative for implementation is to compile the source language into machine language. Such an approach causes the compiler developed to become machine architecture/operating system dependent. In an effort to minimize the dependency risks, the

compiler may actually be developed to translate from the source language to a language already native to the host (hopefully the target language will be portable across host computers).

4.2.1.1 Advantages

Translating the source code to machine code (either directly or through the use of some intermediate high-level language) has several distinct advantages. These advantages are discussed in the following sections.

4.2.1.1.1 Implementation Simplicity

The compilation process from a source language to a target machine language is both mature and well understood. The techniques that would be required to implement the language could be based on proven software concepts.

The approach of translating from one language to another is a well understood and applied principle. Essentially, the compiler has a lookup table which maps symbols between the two languages. Once the original source program is converted to a target program the target program is compiled using an existing compiler. This approach places the overhead on the flight certified compiler to translate the source language to a target language before actually compiling the program.

4.2.1.1.2 Performance

No matter what approach is utilized to implement the flight certified compiler, a performance penalty will be incurred to translate the original source language into some target language. The approach of taking the original source language to machine language will provide an execution speed enhancement over any other alternative. Clearly, a target program in the form of machine code will execute optimally.

The resulting machine code should be optimized to execute in the native implementation environment so that the program will operate at peak efficiency.

4.2.1.2 Disadvantages

The machine code generation implementation strategy has several drawbacks which need to be evaluated. These drawbacks are discussed in the following sections.

4.2.1.2.1 Language Restrictions

A disadvantage to the direct machine code generation process is that the compiler implemented becomes hardware/operating system dependent. As new versions of the operating system, compiler's native language (i.e., the implementing language of the compiler) and support hardware become available, the flight certified compiler would have to be carefully retested to assure that functionality is the same.

A disadvantage to the translation approach to another high order language is that the host language is limited to the functionality that exists within the target language. That is, the statements of the user input language (the source) will need to closely parallel program structures available within the target language.

4.2.1.2.2 Limited Value Added

If the desired implementation strategy was to simply translate the source language into a host defined language (and then use a COTS compiler to generate machine code), an argument would exist as to why not have the user use the COTS language to begin with. Such an argument would have merit because if we are only "front-ending" a language, this can typically be performed through the use of programming macros. A front-end editor could be developed which would provide a structured program.

4.2.1.2.3 Error Recovery

A distinct disadvantage of the machine code generation approach is the uncertainty of what the execution environment will do in the event of catastrophic program failure. In the event that a catastrophic error occurs, determining what actions should be performed is not always possible. In some instances, a catastrophic failure will abort the entire computer or, in many cases, at least the controlling software.

4.2.2 Table Driven Generation

An alternative approach would be to utilize a table driven execution environment. The table driven approach is a variant of an interpreted language, the primary difference being that in a table driven approach, all syntax analysis and variable binding is done before a program is "termed executable." However, in a table driven approach, the source program is represented in an internal table structure (similar to a parse tree), rather than machine code, and an execution executive exists which interprets the table entries during run-time in order to control the execution of the program.

Table-driven execution environments are not traditionally utilized in the implementation of typical higher order languages; rather, they are a useful implementation technique used with object oriented languages. They provide

the capability to perform upfront syntax analysis as well as variable location binding but do not perform the actual binding until run-time.

4.2.2.1 Advantages

The table driven approach to program execution has several advantages over the traditional generation of machine code. The advantages over the traditional generation of machine language are discussed in the following sections.

4.2.2.1.1 Error Recovery

In a table driven environment, the execution environment typically will perform many boundary analysis operations before a segment of code is allowed to execute. The execution environment should not allow a program to get into a state where abnormal termination would occur; rather, the environment would alert the user and prohibit the code from entering an "unsafe" state.

4.2.2.1.2 Semantic Analysis

One of the greater dilemmas facing the development of a flight certified compiler is the one of "How can I make the code do what I want rather than what I told it to do." Of course, this di-

lemma has plagued all language implementors for years. The table driven approach to implementing the flight certified compiler provides some framework to provide semantic analysis of the code before it is actually executed. In a traditional compiler, where machine code is generated from a source language, the resulting machine code should perform exactly what the programmer requested, no more or no less. Such an environment is not conducive to run time semantic analysis of a program.

The execution environment can be made "intelligent" with imposed operational guidelines which are continuously reviewed before actual execution of a statement begins. The execution environment would necessarily be slowed by the introduction of "computational rules" but the concept would provide a mechanism for NASA to place some type of software checks without actually having to modify a piece of source code explicitly.

4.2.2.1.3 Interactive Debugging

The table-driven approach provides the capabilities to provide a rich debugging environment. The execution of the software can be closely associated with specific source commands. As a result, providing informative diagnostics to the user is assured, as well as more accurate diagnostics.

4.2.2.1.4 Portability

Once a program has been converted to a table notation, the table can be ported to a variety of execution environments (e.g., UNIX, MVS, VM) and executed. This will allow flexibility for the development environment to utilize a variety of computer platforms and move code from machine to machine transparently.

4.2.2.2 Disadvantages

The table driven approach also has several disadvantages when compared to the machine code generation approach. These disadvantages are discussed in the following sections.

4.2.2.2.1 Performance

Since the actual execution environment is essentially an interpretive environment, execution performance will not match that of the machine code environment. If a significant number of "operational rules" are embedded into the execution environment, execution performance will degrade. A careful balance of necessary "operational rules" will need to be applied against performance requirements to determine the optimal mix.

4.3 Implementation Considerations

No matter which approach is to be utilized to implement the compiler, a good and robust parser will need to be developed which can provide a complete syntax analysis of the source code. Since the flight certified compiler is targeted for a nonprogramming user, the error reporting from the syntax analysis phase should be robust and helpful. The parser should pinpoint particular tokens (not simply lines of code) which cause the anomaly.

Additionally, since the trend in computer software technology is to lean toward CASE (Computer Aided Software Engineering), the flight certified compiler should be developed in conjunction with a friendly and robust user interface which provides a simplified programming approach. Such interfaces exist within various computing environments and have proven to

be very useful tools. A design decision will have to be made as to whether or not dynamix syntax analysis should be performed (syntax analysis as the program is entered).

The working environment developed for the flight certified compiler should be a highly structured and restricted environment (i.e., the user will not be able to call a module previously defined in another language) so that the goal of developing a safe and reproducible environment can be achieved. While the final environment may not be appealing to accomplished high level programmers, it will be an environment which would allow novice programmers (although operationally proficient) the ability to rapidly develop usable software programs.

4.4 Conclusions

In order to develop a tool which provides maximum flexibility and feedback to the user, the desired approach would be to use a table driven execution environment. While such an environment imposes a small execution penalty, the benefits of dynamic semantic analysis and error recovery far out weigh the performance impact. Additionally, a table driven approach to implementation would provide a tool with maximum flexibility, which is necessary due to the dynamic software development environment present at NASA.

5.0 Verification

The most complicated and involved phase of the flight certified compiler is that of verification. The verification of the compiler will be intense and expensive to perform; however, having the availability of the flight certified compiler will be cost effective in the long run, because each individually developed program will not have to be certified (i.e., the price of verification is being paid for up front rather than over time).

5.1 Grammar Specification Verification

The first major step of the certification process would be to certify that the grammar developed is unambiguous in nature. The grammar has to be implemented in a fashion that allows all flight certified criteria mentioned in section 2.2 to be met.

The grammar which is decreed to be the final language to be implemented will need to be rigorously reviewed in order to assure that no ambiguities, recursive definitions and statements impossible to realize exist within the grammar specification. This process can be performed by defining the language utilizing traditional BNF specification techniques and then determining if the language is an LR grammar. If so, the grammar specification will meet the criteria that the same input always generates the same output (no ambiguous interpretation of the input data stream).

5.2 Language Implementation Verification

The most difficult phase in the development of the flight certified compiler is the actual verification that the implementation is complete and not suspect to generating erroneous results.

The testing of the resulting flight certified compiler needs to be through a good demonstration of software metrics. A rigorous test plan will need to be developed which is capable of performing extensive corner case analysis of each construct associated with the flight certified language. Utilizing existing technology, the testing process will not be highly automated. While it is possible to utilize a computer in assisting with verifying that all possible corner cases are tested, a human being will be required to assure that the translation from specification to actual execution was performed correctly.

It should be noted that implementation verification only demonstrates that the language behaves as designed, it does not provide assurances that user developed programs are correct. While this comment may externally seem to minimize the usefulness of the flight certified compiler, it is actually a strong statement in support; that is, if the language is designed with constructs that will not allow a user to develop applications that will abnormally effect the system, we are assured that all programs written perform exactly what the user directed it to do (i.e., the program can be proven correct).

6.0 Conclusions

The flexibility that a flight certified compiler would provide NASA is worth the effort of continued research. Automating the software certification process will simplify the software development process and will simultaneously allow programs to be developed rapidly as the need arises. Providing a flight controller with the capability to efficiently and reliably generate programs would significantly reduce the typical program development costs which are currently incurred. Additionally, the functionality lost when a flight controller "translates" requirements to a programmer would be minimized.

Additionally, as the Space Station becomes a reality, the practice of having all software completed and certified prior to flight will become difficult. In order for the ground systems to support multi-year flights, they must be provided with the flexibility to generate programs as situations require.

The development of a flight certified compiler should be a task which will not push technology to any unknown limits; the development would primarily be the application of well understood language and compiler techniques. The major limitation of the flight certified compiler is that the compiler can only verify the syntax of the program and assure that the target code generated will not adversely effect the performance of the host computer.

The semantic implication of the program cannot be controlled programmatically; therefore, if an operator devises a program that computes an erroneous result, the flight certified compiler will generate code that reflects that design. However, with the introduction of an intelligent execution environment, partial semantic analysis of a program could be performed in an effort to reduce the incidence of semantic errors.

7.0 Bibliography

- Aho, A., Sethi, R., and Ullman, J., **Compilers Principles, Techniques and Tools**, Addison Wesley, 1985.
- Barrett, W., Bates, R., Gustafson, D., and Couch, J., **Compiler Construction, Theory and Practics**, Science Research Associates, Inc., 1979.
- Bavel, Z., **Introduction to the Theory of Automata**, Reston, 1983.
- Hopcroft, J. and Ullman, J., **Introduction to Automata Theory, Languages, and Computation**, Addison Wesley, 1979.
- Pratt, T., **Programming Languages**, Prentice Hall, 1986.
- Springer - Verlag, **Automata, Languages and Programming**, Springer-Verlag, 1983.

90/06/07
16:14:19

archive.c

1

ARCHIVE

Purpose: If the choice is to create then all of the files associated with a group are written to a formatted floppy.

CodeGroups/GroupName/*
AMSupport/GroupName.dat

If the choice is to read, then first the algo_names.dat file is checked to see if that group already exists. If it exists the user is asked if they want to continue. If it doesn't exist, then it is added to the algo_names.dat file and the all of the files are read into their directories.

If the choice is to list, the files on the floppy are listed in the work area.

Designer: Terri Murphy

Programmer: Terri Murphy

Date: 11/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by: Terri Murphy

Reasons for Revision: Added capability for formatting a floppy and aborting. 12/18/87

External Interfaces

Subroutines:

```
clearWA() -- clears work area window
displayWA() -- display text string in work area window
inform() -- display text in the message window
ask() -- inform(), then prompt for reply
getwd() -- gets the working directory
*****/
```

```
*****
Include files
*****/
#include "code.h"
#include <termio.h> /* used for setting up the keyboard */

struct termio orig_tty; /* used for setting up the keyboard */
struct termio raw_tty; /* used for setting up the keyboard */

archive()
{
    char group[20], /* Name read from GroupNames file */
        sys_cmd[500], /* Used with system calls */
        response[10], /* Used with ask() */
        found,
        abs_group_name[PATH_LEN], /* Name with full path */
        grp_name[20], /* Name without path */
        message[200], /* A place to build string messages */
        workDir[PATH_LEN]; /* A string to hold current work dir. */

    int i, j, /* counters */
        rc; /* return code from function call */

    FILE *tmpfile,
        *grpfile,
        *openFile(); /* A handy file opening routine */

    /*
     * Get the working directory, return to calling routine if
     * unsuccessful.
     */
}
```

```
if (!(getwd(workDir)))
{
    sprintf(message, "Backup could not get the working directory because:\n%s--Hit [RETURN] to continue", wor
kDir);
    ask(message, RED, response);
    return(ERROR);
}

ask("Do you want to (C)reate, (R)ead, (L)ist a floppy, or just (Q)uit this?", GREEN, response);

if (response[0] == 'c' || response[0] == 'C')
{
    ask("Do you need to format a disk (Y/N), or just (Q)uit this?", GREEN, response);

    /*
     * Let's leave unless they answer yes, otherwise let's get on
     * with formatting the floppy disk.
     */
    if(response[0] == 'q' && response[0] == 'Q')
        return ERROR;
    else if (response[0] == 'y' || response[0] == 'Y')
    {
        ask("Place disk to format in drive and press <RETURN> when ready, or (Q)uit this", GREEN, response);
        if(response[0] == 'q' || response[0] == 'Q')
            return ERROR;
        sprintf(sys_cmd, "/etc/flpformat /dev/rflp 2>>/tmp/code.err");
        /*
         * Return to cooked mode to reply to system prompt
         */
        ioctl (fileno (stdout), TCSETAW, &orig_tty);
        ioctl (fileno (stdout), TCFLSH, 0);
        system(sys_cmd);
        ioctl (fileno (stdout), TCSETAW, &raw_tty);
    }

    ask("Place a formatted floppy in the disk drive.\nPress <RETURN> when ready, or just (Q)uit this", GREEN, r
esponse);

    /*
     * If they want to quit, let them.
     */
    if(response[0] == 'q' || response[0] == 'Q')
        return ERROR;
    /*
     * Get the name of the group to copy to floppy.
     */
    if((rc=get_name(NumOfGroups, "Group", GroupName, &GroupNumber) == ERROR))
        return(ERROR);
    /*
     * CREATE to floppy of the group
     */
    inform ("Transferring files to floppy ...", PURPLE, 0);
    clearWA();
    displayWA("\nFILES TRANSFERRED TO FLOPPY:\n\n");
    chdir(CodeGroups);
    sprintf (sys_cmd, "tar cvf /dev/rflp %s/*.h %s/*.v", GroupName, GroupName);
    system(sys_cmd);
    chdir(AMSupport);
    sprintf (sys_cmd, "tar rvf /dev/rflp %s.dat", GroupName);
    if (system(sys_cmd) != ERROR)
    {
        ask("Press <RETURN> to continue", GREEN, response);
    }
    else
    {
        ask("Backup unsuccessful. Hit [RETURN] to continue.", GREEN, response);
    }
    clearWA();
    displayWA(Comp);
}
else if (response[0] == 'r' || response[0] == 'R')
{
    inform ("Reading floppy ...", PURPLE, 0);
    /*
     * If they want to read a floppy, let's first find
     * out what group is on the floppy.
     */
    system ("tar tf /dev/rflp >/tmp/tart 2>>/tmp/code.err");
}
```

```
if (!(tmpfile = openFile ("/tmp/tart", "r", "archive")))
    return (ERROR);
fscanf (tmpfile, "%s", abs_group_name);
fclose (tmpfile);
system ("rm tmpfile 2>>/tmp/code.err");

/*****
Since what we have is the groupName/compNames
and we just want the group name, let's shorten
it up.
*****/
for (i=0, j=0; abs_group_name[i] != '/'; i++, j++)
    grp_name[j] = abs_group_name[i];
grp_name[j] = '\0';

/*****
Let's open the GroupNames file, and see if the
group already exists.
*****/
if (!(grpfile = openFile (GroupNamesFile, "r", "archive")))
    return (ERROR);

while (fscanf (grpfile, "%s", group) != EOF)
{
    if (strcmp(grp_name, group) == 0)
    {
        found = TRUE;
        /*****
        If the group already exists, let the user know
        about it and see if they want to proceed.
        *****/
        ask("This group already exists, do you want to clobber it? (Y/N)", GREEN, response);
        if (response[0] == 'n' || response[0] == 'N')
            return ERROR;
        break;
    }
}

fclose (grpfile);

/*
 * Let's read the floppy finally
 */
inform("Reading files from floppy ...", PURPLE, 0);
chdir(CodeGroups);
clearWA();
displayWA("\nFILES READ FROM FLOPPY\n\n\n");
sprintf(sys_cmd, "tar xvf /dev/rflp %s", grp_name);
system (sys_cmd);
chdir(AMSupport);
sprintf(sys_cmd, "tar xvf /dev/rflp %s.dat", grp_name);
system (sys_cmd);
ask("Press <RETURN> to continue", GREEN, response);
clearWA();

/*****
If the group doesn't exist, then add it to
GroupNames file.
*****/
if(!found)
{
    strcpy(GroupInfo[NumOfGroups].name, grp_name);
    GroupInfo[NumOfGroups++].disposition = 1;
    if (!(grpfile = openFile (GroupNamesFile, "a", "archive")))
        return (ERROR);

    fprintf(grpfile, "%s 1", grp_name);
    fclose(grpfile);
}

}
else if (response[0] == 'l' || response[0] == 'L')
{
    inform ("Reading floppy ...", PURPLE, 0);
    clearWA();
    displayWA("\nLISTING OF FILES ON FLOPPY\n\n\n");
    system ("tar tf /dev/rflp 2>>/tmp/code.err");
    ask("Press <RETURN> to continue", GREEN, response);
    clearWA();
}
```

90/06/07
16:14:19

archive.c

4

```
    displayWA(Comp);  
}  
chdir(workDir);
```

90/06/07
16:14:28

background.c

1

BACKGROUND

Purpose: Background draws the background.

Designers: Troy Heindel & Terri Murphy, both of NASA/JSC/MOD

Programmers: Troy Heindel & Terri Murphy, both of NASA/JSC/MOD

Date: 11/28/88

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

*****/

/*****

Include files

*****/

#include "code.h"

background ()

```
{
    mgigf (BoldFont);
    /*****
    Work Area Box
    *****/
    mgihue (BLUE);
    mgrbox (0.1563, 0.1600, 1.0000, 0.9110);
    /*****
    Keyboard inputs box
    *****/
    mgrbox (0.1563, 0.0800, 1.0000, 0.1600);

    /*****
    Message/Prompts box (draw the lines in yellow)
    *****/
    mgrbox (0.1563, 0.0000, 1.0000, 0.0800);

    mgihue (YELLOW);
    mgrl (0.1563, 0.0000, 0.1563, 0.9330); /* ll to ul */
    mgrl (0.1563, 0.9330, 1.0000, 0.9330); /* ul to ur */
    mgrl (1.0000, 0.9330, 1.0000, 0.0000); /* ur to lr */
    mgrl (0.1563, 0.0000, 1.0000, 0.0000); /* lr to ll */

    mgrl (0.1563, 0.9110, 1.0000, 0.9110); /* Work Area Title Line */
    mgrl (0.1563, 0.0800, 1.0000, 0.0800); /* MESSAGES/PROMPTS Title Line */
    mgrl (0.1563, 0.1600, 1.0000, 0.1600); /* KEYBOARD INPUT Title Line */

    /*****
    File
    *****/
    mgihue (PURPLE);
    mgrbox (0.1562, 0.9380, 0.7100, 1.0000);
    mgihue (YELLOW);
    mgrgfs (0.4181, 0.9780, 0, "File");
    mgrl (0.1562, 0.9780, 0.7100, 0.9780); /* Dividing line */
    mgrl (0.1562, 0.9380, 0.1562, 1.0000); /* Border */
    mgrl (0.1562, 1.0000, 0.7100, 1.0000); /* Border */
    mgrl (0.7100, 1.0000, 0.7100, 0.9380); /* Border */
    mgrl (0.7100, 0.9380, 0.1562, 0.9380); /* Border */

    /*****
    Misc... (Just a button, no title)
    *****/
    mgihue (PURPLE);
    mgrbox (0.7137, 0.9380, 0.8100, 1.0000);
    mgihue (YELLOW);
    mgrgfs (0.7400, 0.9780, 0, "Misc");
    mgrl (0.7137, 0.9780, 0.8100, 0.9780); /* Dividing line */
    mgrl (0.7137, 0.9380, 0.7137, 1.0000); /* Border left */
    mgrl (0.7137, 1.0000, 0.8100, 1.0000); /* Border top */
    mgrl (0.8100, 1.0000, 0.8100, 0.9380); /* Border right */
    mgrl (0.7137, 0.9380, 0.8100, 0.9380); /* Border bottom */
}
```

```

/*****
Font (Just a button, no title)
*****/
mgihue (PURPLE);
mgrbox (0.8137, 0.9380, 1.0000, 1.0000);
mgihue (YELLOW);
mgrgfs (0.8600, 0.9780, 0, "Font Size");
mgrl (0.8137, 0.9780, 1.0000, 0.9780); /* Dividing line */
mgrl (0.8137, 0.9380, 0.8137, 1.0000); /* Border left */
mgrl (0.8137, 1.0000, 1.0000, 1.0000); /* Border top */
mgrl (1.0000, 1.0000, 1.0000, 0.9380); /* Border right */
mgrl (0.8137, 0.9380, 1.0000, 0.9380); /* Border bottom */

/*****
Edit Box
*****/
mgihue (PURPLE);
mgrbox (0.0000, 0.9110, 0.1525, 1.0000);
mgihue (YELLOW);
mgrgfs (0.0575, 0.9780, 0, "Edit");
mgrl (0.0000, 0.9780, 0.1525, 0.9780);
mgrl (0.0000, 0.9110, 0.0000, 1.0000);
mgrl (0.0000, 1.0000, 0.1525, 1.0000);
mgrl (0.1525, 1.0000, 0.1525, 0.9110);
mgrl (0.1525, 0.9110, 0.0000, 0.9110);

/*****
Logic Box
*****/
mgihue (PURPLE);
mgrbox (0.0000, 0.7015, 0.1525, 0.9060);
mgihue (YELLOW);
mgrgfs (0.0513, 0.8840, 0, "Logic");
mgrl (0.0000, 0.8840, 0.1525, 0.8840);
mgrl (0.0000, 0.7015, 0.0000, 0.9060);
mgrl (0.0000, 0.9060, 0.1525, 0.9060);
mgrl (0.1525, 0.9060, 0.1525, 0.7015);
mgrl (0.1525, 0.7015, 0.0000, 0.7015);

/*****
Variables Box (MSID Signal Local Number)
*****/
mgihue (PURPLE);
mgrbox (0.0000, 0.5810, 0.1525, 0.6965);
mgihue (YELLOW);
mgrgfs (0.0363, 0.6745, 0, "Variable");
mgrl (0.0000, 0.6745, 0.1525, 0.6745);
mgrl (0.0000, 0.5810, 0.0000, 0.6965);
mgrl (0.0000, 0.6965, 0.1525, 0.6965);
mgrl (0.1525, 0.6965, 0.1525, 0.5810);
mgrl (0.1525, 0.5810, 0.0000, 0.5810);

/*****
Types Box (Short Integer Float Character)
*****/
mgihue (PURPLE);
mgrbox (0.0000, 0.4600, 0.1525, 0.5760);
mgihue (YELLOW);
mgrgfs (0.0563, 0.5540, 0, "Type");
mgrl (0.0000, 0.5540, 0.1525, 0.5540);
mgrl (0.0000, 0.4600, 0.0000, 0.5760);
mgrl (0.0000, 0.5760, 0.1525, 0.5760);
mgrl (0.1525, 0.5760, 0.1525, 0.4600);
mgrl (0.1525, 0.4600, 0.0000, 0.4600);

/*****
Relational Operators (< <= > >= = <>)
*****/
mgihue (PURPLE);
mgrbox (0.0000, 0.3685, 0.1525, 0.4550);
mgihue (YELLOW);
mgrgfs (0.0263, 0.4330, 0, "Relational");
mgrl (0.0000, 0.4330, 0.1525, 0.4330);
mgrl (0.0000, 0.3685, 0.0000, 0.4550);
mgrl (0.0000, 0.4550, 0.1525, 0.4550);
mgrl (0.1525, 0.4550, 0.1525, 0.3685);
mgrl (0.1525, 0.3685, 0.0000, 0.3685);

```

```

/*****
  Actions Box (Set Print Function Comment)
  *****/
mgihue(PURPLE);
mgrbox(0.0000, 0.2770, 0.1525, 0.3635);
mgihue(YELLOW);
mgrgfs(0.0463, 0.3415, 0, "Action");
mgrl(0.0000, 0.3415, 0.1525, 0.3415);
mgrl(0.0000, 0.2770, 0.0000, 0.3635);
mgrl(0.0000, 0.3635, 0.1525, 0.3635);
mgrl(0.1525, 0.3635, 0.1525, 0.2770);
mgrl(0.1525, 0.2770, 0.0000, 0.2770);

/*****
  Mathematical Ops (+ - * / ( ) Power Sqrt Ln Exp)
  *****/
mgihue(PURPLE);
mgrbox(0.0000, 0.0000, 0.1525, 0.2720);
mgihue(YELLOW);
mgrgfs(0.0570, 0.2500, 0, "Math");
mgrl(0.0000, 0.2500, 0.1525, 0.2500);
mgrl(0.0000, 0.0000, 0.0000, 0.2720);
mgrl(0.0000, 0.2720, 0.1525, 0.2720);
mgrl(0.1525, 0.2720, 0.1525, 0.0000);
mgrl(0.1525, 0.0000, 0.0000, 0.0000);

mgigf (Font);

/*****
  Secret Apple

  That night in August, the trees were full, the boughs
  bent and heavy, and the apples -- all but the bright,
  waxy-green Gravensteins -- were a pale green-going-to-pink.
  The grass in the rows between the trees was knee-high;
  there would be one more mowing before the harvest. That
  night there was an owl hooting from the orchard called
  Cock Hill; Candy and Homer also heard a fox bark from the
  orchard called Frying Pan.

  The Cider House Rules
  John Irving
  *****/
}

```


90/06/07
16:14:34

code.c

1

code.c

Purpose: code.c contains the main event loop which handles token processing.

Designer: Troy Heindel & Terri Murphy of NASA/JSC/MOD

Programmer: Troy Heindel & Terri Murphy of NASA/JSC/MOD

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

External Interfaces

Subroutines:

```
initGraphics() --      inits windows,background
init_code() --  initializes CODE globals
cleanExit() --  exits code, restores original stuff
menu() --      loops to get moused token selection
mgigetfb() --  MC: gets the active frame buffer
clearWA() --   clears work area window
displayWA() -- display text string in work area window
inform() --    display text in the message window
ask() --       inform(), then prompt for reply
mgifb() --     MC: select a frame buffer
put_if() --     puts the 'if' token
put_endif() --  puts the 'endif' token into work area
put_msld() --   put an msld token into work area
put_signal() -- put a signal variable token into work area
put_local() --  put a local variable token into work area
put_lt() --     put a '<' token into work area
put_gt() --     put a '>' token into work area
put_le() --     put a '<=' token into work area
put_ge() --     put a '>=' token into work area
put_ne() --     put a '!=' token into work area
put_eq() --     put a '=' token into work area
put_add() --    put an '+' token into work area
put_subtract() -- put an '-' token into work area
put_multiply() -- put an '*' token into work area
put_divide() --  put an '/' token into work area
put_shiftL() --  put an 'shiftL' token into work area
put_shiftR() --  put an 'shiftR' token into work area
put_l_paren() -- put a '(' token into work area
put_r_paren() -- put a ')' token into work area
put_and() --     put an 'and' token into work area
put_or() --      put an 'or' token into work area
put_bitXor() --  put an 'bitXor' token into work area
put_bitOr() --   put an 'bitOr' token into work area
put_bitAnd() --  put an 'bitAnd' token into work area
put_then() --    put a 'then' token into work area
put_else() --    put an 'else' token into work area
put_not() --     put a 'not' token into work area
put_number() --  put a number into work area
put_string() --  put a string into work area
put_set() --     put a 'set' token into work area
put_func() --    put a math fucntion token into work area
put_comma() --   put a ',' token into work area
save() --        saves the current comp/group to file
edit() --        use vi,ved to edit comp's files
retrieve() --     reads comp from file, resets globals for it
changeFont() --  -- you guessed it!
put_print() --   puts an 'print#' token into work area
get_name() --    prompts user for a comp or group from listing
putStrWithBlue() -- places name on status line in blue bubble
readCompNames() -- reads comp names for a given group
hardcopy() --    prints the current group to printing device
copy() --        copy current comp or group to newly name one
new() --         creates a new comp or group
get_header() --  -- gets comp header info from the user
```

```
delete()    -- deletes the last token from the Comp string
install()   -- installs all completed comps
remove()    -- removes a comp or a group
archive()   -- tars a group to/from floppy
put_pi()    -- put the constant pi into work area
token_help() -- provides token description (modal)
mgiloadcurs() -- MC: loads in a new graphics cursor
color_valid() -- redraws tokens in correct color (yellow, or red)
put_status() -- puts the comp status on status line in bubble

/*****
#include "code.h"
#include "cursor.h"
#include <termio.h>

struct termio orig_tty;    /* used for setting up the keyboard */

main (argc, argv)
    int argc;
    char *argv[];
{
    int    theFont, /* The font that is currently moused */
    theWFont, /* The font last chosen for the WA */
    show_buf, mod_buf, /* frame buffers */
    rc, /* return code from called functions */
    reDraw, /* Boolean whether to redraw comp string */
    oldGroupNumber, /* Group number before retrieve, in case we abort */
    oldCompNumber, /* Comp number before retrieve, in case we abort */
    choice; /* The numeric value of the chosen token */

    char    reply[10], /* A string for user response e.g. Yes/No */
    oldGroupName[PATH_LEN], /* Group name before retrieve, in case we abort */
    oldCompName[PATH_LEN]; /* Comp name before retrieve, in case we abort */

    /*
     * Initialize the windows and graphics.
     */
    initGraphics();

    /*
     * To initialize all flags and strings at start of CODE
     */
    if (init_code() == ERROR)
    {
        /*
         * Make the screen clean. Get cooked. Say bye.
         */
        cleanExit();
    }

    /*
     * The main event loop, run 'till ya' QUIT .
     */
    while(TRUE)
    {
        /*
         * Obtain a token choice from the user
         */
        choice = menu(RUN);
        reDraw = TRUE;

        /*
         * Process the choice
         */
        if (choice == IF)
        {
            put_if ();
        }
        else if (choice == END_IF)
        {
            put_endif ();
        }
        else if (choice == MSID)
        {
            put_msid ();
        }
        else if (choice == SIGNAL)
```

```
(
    put_signal ();
)
else if (choice == LOCAL)
{
    put_local ();
}
else if (choice == LT)
{
    put_lt ();
}
else if (choice == GT)
{
    put_gt ();
}
else if (choice == LE)
{
    put_le ();
}
else if (choice == GE)
{
    put_ge ();
}
else if (choice == NE)
{
    put_ne ();
}
else if (choice == EQ)
{
    put_eq ();
}
else if (choice == ADD)
{
    put_add ();
}
else if (choice == SUBTRACT)
{
    put_subtract ();
}
else if (choice == MULTIPLY)
{
    put_multiply ();
}
else if (choice == DIVIDE)
{
    put_divide ();
}
else if (choice == SHIFTL)
{
    put_shiftL ();
}
else if (choice == SHIFTR)
{
    put_shiftR ();
}
else if (choice == L_PAREN)
{
    put_l_paren ();
}
else if (choice == R_PAREN)
{
    put_r_paren ();
}
else if (choice == AND)
{
    put_and ();
}
else if (choice == OR)
{
    put_or ();
}
else if (choice == BITXOR)
{
    put_bitXor ();
}
else if (choice == BITOR)
{
    put_bitOr ();
}
```

```
}
else if (choice == BITAND)
{
    put_bitAnd ();
}
else if (choice == THEN)
{
    put_then ();
}
else if (choice == ELSE)
{
    put_else ();
}
else if (choice == NOT)
{
    put_not ();
}
else if (choice == NUMBER)
{
    put_number ();
}
else if (choice == STRING)
{
    put_string ();
}
else if (choice == SET)
{
    put_set ();
}
else if (choice == COS)
{
    put_func(COS);
}
else if (choice == ACOS)
{
    put_func(ACOS);
}
else if (choice == SIN)
{
    put_func(SIN);
}
else if (choice == ASIN)
{
    put_func(ASIN);
}
else if (choice == TAN)
{
    put_func(TAN);
}
else if (choice == ATAN)
{
    put_func(ATAN);
}
else if (choice == EXP)
{
    put_func(EXP);
}
else if (choice == LOG)
{
    put_func(LOG);
}
else if (choice == POWER)
{
    put_func(POWER);
}
else if (choice == SQRT)
{
    put_func(SQRT);
}
else if (choice == FUNCTION)
{
    if(put_func(FUNCTION) == ERROR)
    {
        inform("Function executable does not exist",PURPLE,2);
    }
}
else if (choice == COMMA)
{

```

```
    put_comma();
}
else if (choice == EDIT)
{
    /*
     * We save before editing so that the user is editing
     * the latest and greatest files.
     */
    if (NeedToSave)
        save ();
    edit ();

    /*
     * After editing we re-retrieve the modified
     * comp assuming the user has made a modification.
     */
    if (retrieve() == ERROR)
    {
        inform("Unable to retrieve comp after edit",PURPLE,2);
    }
    /*
     * Return to the font they were using.
     */
    changeFont(WORK_AREA, theWAFont);
    reDraw = FALSE;
}
else if (choice == PRINT)
{
    put_print ();
}
else if (choice == COMMENT)
{
    put_comment();
}
else if (choice == SAVE)
{
    save ();
    reDraw = FALSE;
}
else if (choice == RETRIEVE)
{
    /*
     * Give them a chance to save previous work.
     */
    if (NeedToSave)
    {
        ask("Do you want to save your latest work first. (Y/N)",GREEN,reply);
        if (reply[0] == 'Y' || reply[0] == 'y')
            save();
    }

    /*
     * Keep the old name/number around in case the
     * user aborts from the retrieve.
     * Get a group name from the user to retrieve.
     */
    strcpy(oldGroupName,GroupName);
    oldGroupNumber = GroupNumber;
    rc=get_name(NumOfGroups,"Group",GroupName,&GroupNumber);

    if (rc != ERROR)
    {
        /*
         * Keep the comp name and number around in 'old'
         * variables in case the user aborts the get_name.
         * If we do not have an error getting the name, then
         * retrieve the chosen comp.
         */
        strcpy(oldCompName,CompName);
        oldCompNumber = CompNumber;
        rc = get_name(NumOfComps,"Comp",CompName,&CompNumber);
        if(rc != ERROR)
        {
            /*
             * Show us your group name & comp name!
             */
            putStrWithBlue(GroupName, 0.75); /* string, norm x */
            putStrWithBlue(CompName, 0.8823); /* string, norm x */
        }
    }
}
```

```
    /*
     * Read the comp string in from file.
     */
    retrieve ();
}
else /* They aborted the get_name for the comp, so fade back */
{
    strcpy(GroupName,oldGroupName);
    GroupNumber = oldGroupNumber;
    strcpy(CompName,oldCompName);
    CompNumber = oldCompNumber;
    displayWA(Comp);
    reDraw = FALSE;

    /*
     * Read in the comps associated with this group
     */
    readCompNames();
}
else /* The aborted the retrieve, so fall back */
{
    strcpy(GroupName,oldGroupName);
    GroupNumber = oldGroupNumber;
    displayWA(Comp);
    reDraw = FALSE;
}
}
else if (choice == HARDCOPY)
{
    /*
     * We save if necessary since hardcopy prints the files.
     */
    if (NeedToSave) save();
    hardcopy ();
    reDraw = FALSE;
}
else if (choice == COPY)
{
    if (NeedToSave) save ();
    copy ();
}
else if (choice == NEW)
{
    /*
     * Give them a chance to save previous work.
     */
    if(NeedToSave)
    {
        ask("Do you want to save your latest work first. (Y/N)",GREEN,reply);
        if (reply[0] == 'Y' || reply[0] == 'y')
            save();
    }
    /*
     * If there are zero groups then they must want a new group
     * since comps must be associated with a group.
     */
    if(NumOfGroups<1)
        reply[0] = 'G';
    else ask("Would you like a new (G)roup, (C)omp, or (Q)uit this?",GREEN,reply);
    if(reply[0] == 'G' || reply[0] == 'g')
    {
        if(new("Group")!=DEFAULT) /* Get the group, then */
        {
            if(new("Comp")!=DEFAULT) /* the comp.*/
                get_header(); /* We need a header for this new guy */
        }
    }
    else if (reply[0] == 'c' || reply[0] == 'C')
    {
        /*
         * Get the name of a group from the user to put this
         * new comp in.
         */
        rc=get_name(NumOfGroups,"Group",GroupName,&GroupNumber);

        if (rc != ERROR)
```

```
{
    /*
     * Show us your group name!
     */
    putStrWithBlue(GroupName, 0.75); /* string, norm x */

    /*
     * Read in the comps associated with this group
     */
    readCompNames();
}

/*
 * Create the new comp.
 */
if (new("Comp") != DEFAULT)
    get_header(); /* We need a header for this new guy */
)
else reDraw = FALSE;
}
else if (choice == DELETE)
{
    delete ();
    /*
     * You must clean after a delete to see what happened.
     */
    clearWA();

    /*
     * If we're down to the last token then start the comp
     * over again by getting the header.
     */
    if (ChoiceCounter < 1)
        get_header ();
}
else if (choice == INSTALL)
{
    if (NeedToSave) save ();
    install ();
}
else if (choice == REMOVE)
{
    /*
     * Give them a chance to save previous work.
     */
    if (NeedToSave)
    {
        ask("Do you want to save your latest work first. (Y/N)", GREEN, reply);
        if (reply[0] == 'Y' || reply[0] == 'y')
            save();
    }
    remove ();
    clearWA();
    reDraw = FALSE;
}
else if (choice == BACKUP)
{
    if (NeedToSave) save ();
    archive ();
    reDraw = FALSE;
}
else if (choice == PI)
    put_pi ();
else if (choice == QUIT)
{
    ask("Are you sure you want to quit?", GREEN, reply);

    if (!(reply[0] == 'N' || reply[0] == 'n'))
    {
        if (NeedToSave)
        {
            ask("Save this comp first? (Y/N)", GREEN, reply);
            if (reply[0] == 'Y' || reply[0] == 'y')
                save();
        }
        cleanExit();
    }
}
}
```

90/06/07
16:14:34

code.c

8

```
else if (choice == HELP)
{
    mgiloadcurs (7,7,4,Help);
    inform("You are now in the HELP mode",PURPLE,0);
    choice = menu(HELP);
    token_help(choice, HELP);
    mgiloadcurs (7,7,4,Shuttle);
    reDraw = FALSE;
}
else if (choice == LIST)
{
    mgiloadcurs (7,7,4,List);
    inform("You are now in the LIST mode",PURPLE,0);
    choice = menu(LIST);
    token_help(choice, LIST);
    mgiloadcurs (7,7,4,Shuttle);
    reDraw = FALSE;
}
else if (choice == SMALL || choice == MEDIUM || choice == LARGE)
{
    theFont = choice;
    inform("You are now in the FONT mode",PURPLE,0);
    mgiloadcurs (7,7,4,FontCurs);
    choice = menu(FONT);
    if(choice == WORK_AREA)
    {
        theWAFont = theFont;
        changeFont(WORK_AREA, theFont);
    }
    else changeFont(ERROR, theFont);
    mgiloadcurs (7,7,4,Shuttle);
    clearWA();
}

/*
 * The next little piece of code sets up the next
 * available choices and positions the mouse.
 */
if(reDraw||choice!=ERROR)
{
    color_valid (PrevChoice[ChoiceCounter - 1], choice);
    displayWA(Comp);
    put_status ();
}
inform("",BLUE,0);
}
```


90/06/07
16:22:01

code.h

1

```
#include "color.h"
#include <stdio.h>
#include <math.h>
#include <signal.h>
#include <sys/file.h>
#include <sys/ioctl.h>
/*****
GLOBAL CONSTANTS
*****/
#define PATH_LEN 80 /* The number of chars allocated for the string.*/
#define MAX_NAME_LEN 14 /* The number of chars allocated for any file name. */
#define MAX_COMPS 100
#define MAX_GROUPS 100
#define NOMEN_LEN 50
#define MAX_VAR_LEN 30
#define TYPE_LEN 10
#define MAX_COMP_VARS 50
#define MAX_GROUP_VARS 500
#define MAX_COMP_LEN 20000
#define MAX_TOKENS 2000
#define MSID_NAME_LEN 12 /* The number of chars allocated for an MSID name */
#define SIGNAL_NAME_LEN 21 /* The number of chars allocated for an MSID name */
#define MAX_NUM_MSIDS 4000 /* The max number of MSIDs which can be defined in tag tbl */
#define MAX_NUM_SIGNALS 800 /* The max number of SIGNALs which can be defined in signal tbl */
#define MAX_SETTERS 50 /* The maximum number of comps which are allowed to set a signal */
#define INCOMPLETE 0 /* Used to describe a comps disposition */
#define COMPLETE 1 /* Used to describe a comps disposition */
#define INSTALLED 2 /* Used to describe a comps disposition */
#define PUT 0 /* Used to describe a how a signal is being used */
#define GET 1 /* Dito above */
#define PET 2 /* Dito above */
#define ERROR -1 /* What can I say, sometimes it happens */
#define DEFAULT 1
#define OK 0 /* What can I say, sometimes it happens */
#define GROUP_ONLY 0 /* Argument to get_a_name - Get group name only */
#define GROUP_COMP 1 /* Argument to get_a_name - Get group and comp name */
#define MAX_NESTED_IF 12 /* The max number of nested if's we allow the user */
#define MAX_COMPARES 100 /* The max number of variable comparisons in a comp */
#define MAX_STR_LEN 120 /* The max length of a fault msg, or SIGNAL string */
#define MAX_MSG_LEN 300 /* The max length of a prompt area string to the user */
#define FUNC_NAME_LEN 11 /* The max number of characters in a user function name */
#define MAX_FUNC_PARAMS 20 /* The max number of parameters in a user defined function */
#define MAX_NUM_FUNCS 100 /* The max number of user functions which can be stored */
#define TOKEN_SIZE 500 /* The max length of a token string */
#define DELETING 0 /* Used to tell get_a_name we are getting something to delete */
#define ARCHIVING 1 /* Used to tell get_a_name we are getting something to archive */
#define RETRIEVING 2 /* Used to tell get_a_name we are getting something to retrieve */
#define Font 1 /* Used in code.c */
#define BoldFont 2 /* Used in code.c */
#define ItalicFont 3 /* Used in code.c */
#define HugeFont 4 /* Used in code.c */
#define SmallTextFont 2 /* Used in code.c */
#define MediumTextFont 0 /* Used in code.c */
#define LargeTextFont 1 /* Used in code.c */
#define SIZE_INDENT 5 /* The size of nominal indent */
#define FIRST_SIGNAL "SIGNAL_ZERO" /* Used in retrieve to determine when the header's been ripped */
#define MAX_COMMENT_LEN 250 /* Used in put_status to determine if comp length is getting close to max */
#define MORE "Hit => SpaceBar:Advance Page RETURN:Advance Line Q:Quit Window h:Help"

#define TRUE 1
#define FALSE 0

#define PROMPT 2
#define INFO 1
#define NO_CHANGE 0

#define PREMISE 0
#define CONSEQUENCE 1

#define LHS 0
#define RHS 1

#define NUMBER_OF_OPTIONS 66
#define NUMBER_OF_HELP 3

#define dead_end -1
#define IF 0
#define THEN 1
```

30/06/07
16:22:01

code.h

2

```
#define AND 2
#define OR 3
#define NOT 4
#define PRINT 5
#define SET 6
#define GT 7
#define GE 8
#define LT 9
#define LE 10
#define EQ 11
#define MSID 12
#define SIGNAL 13
#define LOCAL 14
#define NUMBER 15
#define INTEGER 16
#define FLOAT 17
#define DOUBLE 18
#define NEW 19
#define SAVE 20
#define INSTALL 21
#define RETRIEVE 22
#define QUIT 23
#define DELETE 24
#define ELSE 25
#define END_IF 26
#define LIST 27
#define NE 28
#define L_PAREN 29
#define R_PAREN 30
#define ADD 31
#define SUBTRACT 32
#define MULTIPLY 33
#define DIVIDE 34
#define REMOVE 35
#define EDIT 36
#define COMMENT 37
#define HARDCOPY 38
#define BITXOR 39
#define BACKUP 40
#define SHORT 41
#define CHAR 42 /* A type */
#define FUNCTION 43
#define EXP 44
#define LOG 45
#define COS 46
#define ACOS 47
#define SIN 48
#define ASIN 49
#define TAN 50
#define ATAN 51
#define COPY 52
#define SQRT 53
#define POWER 54
#define STRING 55 /* Looks like "I am a string" */
#define BITAND 56
#define BITOR 57
#define SHIFTL 58
#define SHIFTR 59
#define PI 60
#define COMMA 61
#define HELP 62
#define SMALL 63
#define MEDIUM 64
#define LARGE 65

#define WORK_AREA 66
#define KEYIN_AREA 67
#define PROMPT_AREA 68
#define EDIT_WINDOW 69

#define FONT 998
#define RUN 999

#define GET_TYPE 1001
/*****
The group_list_struct is used to store the listing of group names
and dispositions read from the GroupNames file.
*****/
```

```
struct group_info_struct
{
    char name[MAX_NAME_LEN]; /* The names of the groups read from GroupNames file*/
    int disposition; /*1 = error, 0 = incomplete, 1 = complete, 2= installed */
};

struct group_info_struct GroupInfo[MAX_GROUPS];

/*****
The group_info_struct is used to store the data pertaining to the group
which is found in the <group_name>.dat file. The data stored in this
file is also used by the Algorithm Manager.
*****/

struct comp_info_struct
{
    char name[MAX_NAME_LEN];
    int disposition; /* -1 = error, 0 = incomplete, 1 = complete, 2= installed */
    int noise_filter; /* integer number 1-9 */
    int rate; /* integer number 1-9 */
    int on_off; /* 0 = Off, 1 = On */
    char purpose[80];
};

struct comp_info_struct CompInfo[MAX_COMPS];

/*****
The var_struct is used in an array that contains all of the
variables for the group, and another array that contains all of the
variables for the current comp.
*****/

struct var_struct
{
    char name[MAX_VAR_LEN];
    char type [TYPE_LEN];
    char class [TYPE_LEN];
    int occurrence;
    int put_or_get;
    float lo1_limit; /* ops low limit */
    float lo2_limit; /* critical low limit */
    float hi1_limit; /* ops high limit */
    float hi2_limit; /* critical high limit */
    char nomenclature[NOMEN_LEN];
};

struct var_struct CompVars[MAX_COMP_VARS];
struct var_struct GroupVars[MAX_GROUP_VARS];

/*****
The msid_tbl_struct is use dthe store the tag msid table which is
kept in the file ../tables/tag_msid_tbl. It is a list of the
defined msids which will have tlm.
*****/

struct msid_tbl_struct
{
    char name[MSID_NAME_LEN];
    char type[TYPE_LEN];
    char nomenclature[NOMEN_LEN];
};

struct msid_tbl_struct MSIDTable[MAX_NUM_MSIDS];

/*****
The sig_tbl_struct is used to store the signal table information
which is kept in the file /user/tables/signal_table.
*****/

struct sig_tbl_struct
{
    char name[SIGNAL_NAME_LEN];
    char type[TYPE_LEN];
    int putter_count; /* The number of comps which put this signal */
    char putter[MAX_SETTERS][MAX_NAME_LEN + MAX_NAME_LEN + 2]; /* The names of the putters */
    int getter_count; /* The number of comps which set this signal */
    char getter[MAX_SETTERS][MAX_NAME_LEN + MAX_NAME_LEN + 2]; /* The names of the setters */
    char nomenclature[NOMEN_LEN];
};
```

```
struct sig_tbl_struct SignalTable[MAX_NUM_SIGNALS];
/*****
GLOBAL VARIABLES
*****/
char  CompName[MAX_NAME_LEN], /* Used in nocip, get_a_name, save, retrieve. */
      GroupName[MAX_NAME_LEN], /* The name of the group in Work Area */
      UserFuncs[MAX_NUM_FUNCS][FUNC_NAME_LEN], /* For user function name storage */
      Comp[MAX_COMP_LEN], /* The all important string for holding the CODE language tokens */
      CompareType[MAX_COMPARES][TYPE_LEN]; /* Stores the type of each of the var. compares */
      SignalTbl[PATH_LEN], /* Set in init_code, used throughout */
      MSIDTbl[PATH_LEN], /* Set in init_code, used throughout */
      CodeGroups[PATH_LEN], /* Set in init_code, used throughout */
      GroupNamesFile[PATH_LEN], /* Set in init_code, store path for file of group names */
      CodeDocs[PATH_LEN], /* The path with file for the code documentation */
      Code[PATH_LEN], /* Set in init_code, used throughout */
      UserFuncsLib[PATH_LEN], /* Set in init_code, used throughout */
      RTDS[PATH_LEN], /* Path to the RTDS system */
      AMSupport[PATH_LEN], /* Set in init_code, used throughout */
      AMGroups[PATH_LEN], /* Set in init_code, used throughout */
      FunctionList[MAX_NUM_FUNCS][FUNC_NAME_LEN];

int   NumCompVars, /* Used in save, retrieve, var_check. The number of variables used in a comp. */
      NumGroupVars, /* Used in save, retrieve. The number of variables used in a group. */
      CompNumber, /* The index of the comp in the group_info structure */
      GroupNumber, /* The index of the group in the group_list structure */
      SignalCount, /* The number of signal read from the signal table */
      MSIDCount, /* The number of msids read from the tag_msid_tbl */
      NumOfComps, /* The number of comps in the active group */
      NumOfGroups, /* The number of groups */
      NumberOfIfs, /* The number of ifs in the work area comp */
      NumberOfEndifs, /* The number of endifs in the work area comp */
      NumberOfCompares, /* The number of equal signs in a comp, both EQ, and EQEQ */
      NestedElseCheck[MAX_NESTED_IF], /* Used in next_inputs to determine if else is valid */
      NumberOfUserFuncs, /* the number of user funcs read into init_code.c */
      NeedToSave, /* TRUE means we have made changes to the WA comp */
      Equation, /* LHS or RHS */
      ParenCount, /* The number of parenthesis */
      TotValPts, /* The number of tokens which are valid; menu() and next_inputs() */
      ValidPoints[NUMBER_OF_OPTIONS], /* The list of valid tokens; menu() and next_inputs() */
      LikelyNextChoice, /* The token to position the mouse over; menu() and next_inputs() */
      WhereAmI, /* Either PREMISE or CONSEQUENCE */
      ChoiceCounter, /* An index into the PrevChoice array of tokens */
      PrevChoice[MAX_TOKENS], /* An integer history of the tokens in a comp */
      FunctionCount, /* The number of functions used in a comp */
      FunctionArguments[MAX_NUM_FUNCS], /* The number or arguments needed to satisfy the function */
      FunctionArgsDef[MAX_NUM_FUNCS], /* The number or arguments entered thus far */
      FunctionCurrent, /* The function we are currently trying to satisfy arguments for */
      FuncParenCount, /* The paren count for function argument definitions */
      ColorMap[32]; /* array to store and restore the color map */
```

90/06/07
16:12:14

color.h

1

```
#define BLACK      0
#define BLUE       1
#define RED        2
#define ORANGE     4
#define YELLOW     8
#define PURPLE    12
#define GREEN     16
#define OTHER_WHITE 18
#define WHITE     18
#define BROWN     20

#define background_color    PURPLE
#define text_color          YELLOW
#define border_color        RED
#define box_color           YELLOW
```

90/06/07
16:22:19

copy.c

1

COPY

Purpose: Copy copies a comp or a group into a new comp or group.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/20/89

Version: 2.0

Project: CODE (Comp Development Environment)

External Interfaces

```
ask()          -- prompts the user for a string
new()          -- creates a new comp or group
putStrWithBlue() -- puts the name on status line in bubble
save()         -- saves the current comp/group to disk
switch_name()  -- changes the comp name in its header
*****/
#include "code.h"
```

```
copy ()
{
    char reply[200],          /* a temp string */
        oldName[MAX_NAME_LEN], /* a place to hold the new name */
        pathOldGroup[PATH_LEN], /* the path for the old group */
        pathNewGroup[PATH_LEN]; /* the path for the new group */

    int oldNumber;           /* Used to store the number of comps */

    /*
     * See if they want to copy a comp or a group
     */
    if(ask("Make copy of this (C)omp, (G)roup, or just (Q)uit this?",GREEN,reply)==DEFAULT||
        reply[0] == 'Q' || reply[0] == 'q')
        return (ERROR);
    if(reply[0] == 'G' || reply[0] == 'g')
    {
        /*
         * First off, let's build some paths to the source
         * groups files, so we can copy them later on into
         * the destination groups files
         */
        sprintf(pathOldGroup,"%s/%s",CodeGroups,GroupName);

        /*
         * Get the name of the comp the user wishes
         * to copy from from get_new_name.
         */
        strcpy(oldName, GroupName);
        oldNumber = GroupNumber;
        if (get_new_name("Group", GroupName))
            return;

        /*
         * Copy the comps from the old group into the new one.
         * Note that GroupName was modified in get_new_name()!
         */
        sprintf(pathNewGroup,"%s/%s",CodeGroups,GroupName);
        if(mkdir (pathNewGroup, 511))
        {
            ask("CODE is unable to make the new directory - copy aborted.\nHit [RETURN] to continue",RED,reply);
            strcpy (GroupName, oldName); /* Undo the damage */
            return (ERROR);
        }
        inform ("Copying ...", PURPLE);
        sprintf(reply,"cp %s/* %s>/tmp/tart 2>>/tmp/code.err",pathOldGroup,pathNewGroup);

        /*
         * If we fail to copy the files we should go back to the
         * original configuration.
         */
        if(system(reply)!=0)
        {
            ask("CODE is unable to copy the files into the new directory;copy aborted.\nHit [RETURN] to continue",
,RED,reply);
            strcpy (GroupName, oldName); /* Undo the damage */
            return (ERROR);
        }
        /*
         * Since we had a successful copy, copy the group
         * info from the old group into the new group, and
         * copy the GroupName into the GroupInfo.
         */
        GroupNumber = NumOfGroups++;
        GroupInfo[GroupNumber] = GroupInfo[oldNumber];
        strcpy(GroupInfo[GroupNumber].name, GroupName);
        /*
         * Paste the new group name up on the status line
         */
        putStrWithBlue(GroupName, 0.75); /* string, norm x */
        save();
    }
    /*
     * else they want to copy a COMP
     */
    else if(reply[0] == 'C' || reply[0] == 'c')
    {
        /*

```

30/06/07
16:12:19

copy.c

3

```
* Get the name of the comp the user wishes
* to copy from from new(), but
* remember to keep the old name around for
* use in the call to switch_name().
*/
strcpy(oldName, CompName);
oldNumber = CompNumber;
if (get_new_name("Comp", CompName))
    return;
inform ("Copying ...", PURPLE);
/*
* Switch the old name which is in the comps header
*/
if (switch_name (oldName) == ERROR)
    ask("An error occurred while changing the name.\nHit [RETURN] to continue",RED,reply);
/*
* Copy the old comp info into the new comp info,
* copy the CompName into the CompInfo
* and increment the NumOfComps.
*/
CompNumber = NumOfComps++;
CompInfo[CompNumber] = CompInfo[oldNumber];
strcpy(CompInfo[CompNumber].name, CompName);
/*
* Paste the group & new comp name up on the screen
*/
putStrWithBlue(CompName, 0.8823); /* string, norm x */
putStrWithBlue(Groupname, 0.75); /* string, norm x */

/*
* Save the resulting new comp
*/
save();
inform ("Copy Complete", PURPLE, 2);
clearWA();
}

return (OK);
```


/*****

SWITCH_NAME

Purpose: Switch_name goes through the comp string and changes the old name to the new name. This routine puts the comp into a string until it finds the CompName, then it passes the comp name, and puts the rest of the Comp in another string. When finished doing this, it copies the first part of the comp into Comp, then cats the new comp name into Comp, then finally, cats the second part of the comp.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/8/88
1/20/89 rewritten, TAH

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

*****/

#define MAX_HEAD_LEN 300

switch_name(oldName)

char oldName[];

{

```
    int    i,          /* Index into the Comp string */
    l,      /* Index into onePartOfComp */
    r,      /* Index into otherPartOfComp */
    k,      /* Index into oldName */
    nameLength; /* The length of the comp name */
```

```
    char temp[40], /* Debug */
    onePartOfComp[MAX_HEAD_LEN], /* Comp string before oldName */
    otherPartOfComp[MAX_COMP_LEN], /* Comp string after oldName */
    foundName=FALSE; /* Flag */
```

```
    /*
    * Get the length of the comp name
    */
    nameLength = strlen (oldName);
```

```
    /*
    * Process through the Comp string looking
    * for the old comp name.
    */
```

```
    i = k = l = r = 0;
    while (Comp[i] != 0 && i < MAX_COMP_LEN)
    {
```

```
        /*
        * If we have not yet found the name
        * then continue to look for it.
        */
```

```
        if(!foundName)
        {
```

```
            /*
            * If we don't find the comp name in
            * MAX_HEAD_LEN chars then let's return
            */
            if(i>=MAX_HEAD_LEN)
                return ERROR;
```

```
            /*
            * Add a char from the Comp string to
            * a string called onePartOfComp.
            */
            onePartOfComp[l++] = Comp[i];
```

```
            /*
            * If we find a match, see if its the
            * last char match we need to make.
            */
            if(oldName[k] == Comp[i++])
```

```
(  
    /*  
     * If we have matched nameLength chars then  
     * we have found the old name.  
     */  
    if (++k == nameLength)  
    {  
        onePartOfComp[l-nameLength] = 0; /* terminate */  
        foundName = TRUE;  
    }  
    else k = 0; /* reset the matched letter counter */  
}  
/*  
 * Now that we have found the comp name let us  
 * gather up the rest of the Comp string  
 * into another string.  
 */  
else otherPartOfComp[r++] = Comp[i++];  
}  
  
otherPartOfComp[r] = 0; /* terminate */  
  
/*  
 * Copy the half preceding the old comp name into Comp, then cat the  
 * the new comp name onto that, and add in the otherPart of the comp  
 */  
strcpy (Comp, onePartOfComp);  
strcat (Comp, CompName);  
strcat (Comp, otherPartOfComp);  
}
```

90/06/07
16:22:28

cursor.h

1

```

/*****
This is the shuttle cursor!
*****/
static short Shuttle[16] = {
    0xe000, 0x9000, 0xb800, 0x7c00, 0x3f00, 0x1fc0, 0x0fff, 0x0fff,
    0x077e, 0x07bc, 0x03db, 0x03e2, 0x03ec, 0x03c8, 0x03b2, 0x0321
};
/*****
This is the help cursor!
*****/
static short Help[16] = {
    0xffff, 0x8001, 0x83c1, 0x87e1, 0x8c31, 0x8c31, 0x8061, 0x80c1,
    0x8181, 0x8181, 0x8181, 0x8001, 0x8181, 0x8181, 0x8001, 0xffff
};
/*****
This is the list cursor!
*****/
static short List[16] = {
    0x8eee, 0x84a4, 0x8484, 0x8484, 0x84e4, 0x8424, 0x8424, 0x84a4,
    0xeeee, 0x0000, 0xfffe, 0x0004, 0x0008, 0x0000, 0x0000, 0x0000};
/*****
This is the font cursor!
*****/
static short FontCurs[16] = {
    0xfc20, 0xfc70, 0xc0a8, 0xc020, 0xc020, 0xc020, 0xf820, 0xf820,
    0xc020, 0xc020, 0xc020, 0xc0a8, 0xc070, 0xc020, 0x0000, 0x0000};

```

90/06/07
16:22:39

delete.c

1

DELETE

Purpose: Delete removes the last token in the Comp string.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/19/87

Version: 1.0

Project: CODE (Comp Development Environment)

External Interfaces

getTokenLen() -- returns the length of a given token
dcrVarList() -- decrements the occurrence count for a variable
ask() -- prompts the user for a char string
delete() -- look up bub!
lookBackJack() -- check whether prev paren is preceded by function

*****/
#include "code.h"

```
delete ()
{
    int    compLength,    /* length of the comp */
          tokenLength,    /* length of the token */
          i,              /* index into Comp */
          tokenNumber,    /* the number of the token to delete */
          j;              /* index into var_temp */

    char    var_temp[MAX_VAR_LEN], /* the variable to delete */
           message[169],          /* a message string */
           reply[50];             /* for user reply in ask */

    tokenNumber = PrevChoice [ChoiceCounter - 1];
    tokenLength = getTokenLen(tokenNumber);
    compLength = strlen (Comp);

    /*
     * If the token is a variable we need
     * to find and index to its first char.
     */
    if (tokenNumber == MSID || tokenNumber == SIGNAL || tokenNumber == LOCAL)
    {
        /*
         * Copy the variable into a temp string
         */
        for (j=0,i=compLength-tokenLength+1;i<compLength;i++)
            var_temp[j++] = Comp[i];
        var_temp[j] = 0; /* terminate string */

        /*
         * Decrement the occurrence of this variable
         * or delete if last one.
         */
        dcrVarList (var_temp);
    }
    /*
     * Shorten up the CODE comp string to its new more
     * diminutive stature.
     */
    Comp[compLength - tokenLength] = 0;
    ChoiceCounter--;
    NeedToSave = TRUE;

    /*
     * Reset some global flags & counters
     */
    if(tokenNumber == IF)
    {
        WhereAmI = CONSEQUENCE;
        Equation = RHS;
        NumberOfIfs--;
    }
    else if (tokenNumber == THEN)
    {
        WhereAmI = PREMISE;
        Equation = RHS;
    }
    else if (tokenNumber == AND || tokenNumber == OR ||
             tokenNumber == SET || tokenNumber == ELSE)
    {
        Equation = RHS;
        /*
         * Reset this NestedElseCheck so that ELSE is
         * enabled in next_inputs
         */
        if(tokenNumber == ELSE)
            NestedElseCheck[NumberOfIfs-NumberOfEndifs] = THEN;
    }
    else if(tokenNumber == END_IF)
    {
        NumberOfEndifs--;
        Equation = RHS;
    }
    else if (tokenNumber == R_PAREN)
    {
        ParenCount++;
    }
}
```

```
/*
 * Call to lookBackJack() looks backwards to see
 * whether or not we are inside of a function.
 */
if (FuncParenCount > 0 || lookBackJack ())
    FuncParenCount++;
}
else if (tokenNumber == L_PAREN)
{
    ParenCount--;
    /*
     * If we are working inside of a function expression
     * we need to maintain its paren count as well
     */
    if (FuncParenCount > 0)
        FuncParenCount--;

    /*
     * If the previous choice is a function then delete it
     * also since it is linked to this left paren.
     */
    if (PrevChoice[ChoiceCounter-1] == NOT ||
        PrevChoice[ChoiceCounter-1] == COS ||
        PrevChoice[ChoiceCounter-1] == ACOS ||
        PrevChoice[ChoiceCounter-1] == SIN ||
        PrevChoice[ChoiceCounter-1] == ASIN ||
        PrevChoice[ChoiceCounter-1] == TAN ||
        PrevChoice[ChoiceCounter-1] == ATAN ||
        PrevChoice[ChoiceCounter-1] == POWER ||
        PrevChoice[ChoiceCounter-1] == LOG ||
        PrevChoice[ChoiceCounter-1] == EXP ||
        PrevChoice[ChoiceCounter-1] == FUNCTION ||
        PrevChoice[ChoiceCounter-1] == SQRT)
        delete ();
    }
else if (tokenNumber == EQ ||
        tokenNumber == LT || tokenNumber == GT ||
        tokenNumber == LE || tokenNumber == GE ||
        tokenNumber == NE)
{
    Equation = LHS;
    NumberOfCompares--;
}
}
```

```
/******
    lookBackJack
This is a short routine which looks back into the token history
to see if the paren we are deleting belongs to a function.
*****/
lookBackJack ()
{
    int    parenCount, /* Number of parens we've found so far */
           i;          /* index into PrevChoice */

    i = ChoiceCounter; /* start at the end */
    parenCount = 0;

    /*
     * Move backwards through the token string trying to
     * find the match to the paren we have in our hand.
     */
    do
    {
        if (PrevChoice[i] == R_PAREN)
        {
            parenCount++;
        }
        else if (PrevChoice[i] == L_PAREN)
        {
            parenCount--;
        }
    } while (parenCount > 0 && i-- > 0);

    /*
     * If the matching paren is just after a function,
     * then return TRUE.
     */
    if (PrevChoice[i-1] == NOT    || PrevChoice[i-1] == SQRT ||
        PrevChoice[i-1] == COS   || PrevChoice[i-1] == ACOS ||
        PrevChoice[i-1] == SIN   || PrevChoice[i-1] == ASIN ||
        PrevChoice[i-1] == TAN   || PrevChoice[i-1] == ATAN ||
        PrevChoice[i-1] == LOG    || PrevChoice[i-1] == EXP ||
        PrevChoice[i-1] == POWER || PrevChoice[i-1] == FUNCTION)
        return (TRUE);
    else return (FALSE);
}
```

90/06/07
16-22:39

delete.c

5

```
/******  
dcrVarList
```

Purpose: dcrVarList decrements the occurrence of a passed variable if it finds it in the CompVars structure. It will also remove the variable from the CompVars structure if the occurrence count is 0.

Returns: 0 if success, -1 if variable not found.

Designer: Troy Heindel/NASA

Programmer: Troy Heindel/NASA

Date: 11/20/88

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

```
*****  
dcrVarList (variableName)  
char variableName[];  
{  
    int    i, j, index, /* loop counters, and indexing */  
    signal_index; /* Index of the signal in the signal table */  
  
    /*  
    * Get the index of the passed variable and decrement its  
    * occurrence counter by 1, and remove it if its the last  
    */  
    for (index=0; index<NumCompVars; index++)  
    {  
        if (strcmp (CompVars[index].name, variableName) == 0)  
        {  
            /******  
            If there are now 0 occurrences of this variable we  
            we remove it from the list of comp variables  
            *****/  
            if (--CompVars[index].occurrence == 0)  
            {  
                for (j=index; j<NumCompVars; j++)  
                {  
                    strcpy (CompVars[j].name, CompVars[j+1].name);  
                    strcpy (CompVars[j].type, CompVars[j+1].type);  
                    strcpy (CompVars[j].class, CompVars[j+1].class);  
                    CompVars[j].occurrence = CompVars[j+1].occurrence;  
                    CompVars[j].put_or_get = CompVars[j+1].put_or_get;  
                    CompVars[j].lo1_limit = CompVars[j+1].lo1_limit;  
                    CompVars[j].lo2_limit = CompVars[j+1].lo2_limit;  
                    CompVars[j].hi1_limit = CompVars[j+1].hi1_limit;  
                    CompVars[j].hi2_limit = CompVars[j+1].hi2_limit;  
                    strcpy(CompVars[j].nomenclature, CompVars[j+1].nomenclature);  
                }  
                --NumCompVars;  
            }  
            /******  
            Let's break out since we found it  
            *****/  
            break;  
        }  
    }  
    return;  
}
```



```
*****  
getTokenLen
```

Purpose: getTokenLen takes a token and figures out its string length, which is needed for adding and deleting tokens, and retrieving comps.

Designer: Troy Heindel/NASA

Programmer: Troy Heindel/NASA

Date: 5/29/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/  
getTokenLen (tokenClass)  
    int tokenClass; /* IF, THEN, ...etc... */  
{  
    int tokenLength,  
        compLength,  
        i;  
  
    /*  
     * Get the length of the entire comp string  
     */  
    tokenLength = 0;  
    compLength = strlen (Comp);  
  
    /*  
     * PRINT (case 1)  
     */  
    if(tokenClass == PRINT)  
    {  
        /*  
         * Find both sets of double quotes  
         */  
        for (i=0; i<2; i++)  
        {  
            /*  
             * Move backwards finding double quotes  
             */  
            do  
            {  
                tokenLength++;  
                compLength--;  
            } while (Comp[compLength] != '"' && compLength);  
        }  
  
        /*  
         * Add 8 for ' print# '  
         */  
        compLength -= 8;  
        tokenLength += 8;  
  
        /*  
         * Move backwards finding the previous token  
         */  
        while ((Comp[compLength-1] == ' ' || Comp[compLength-1] == '\n') && compLength)  
        {  
            tokenLength++;  
            compLength--;  
        }  
    }  
  
    /*  
     * COMMENT (case 2)  
     */  
    else if(tokenClass == COMMENT)
```

```
{
    /*
     * Find the start comment delimiter '/*'
     */
    do
    {
        tokenLength++;
        compLength--;
    } while (!(Comp[compLength] == '/' && Comp[compLength+1] == '*') && compLength);
}
/*
 * STRING (case 4)
 */
else if (tokenClass == STRING)
{
    /*
     * Find both sets of double quotes
     */
    for (i=0; i<2; i++)
    {
        /*
         * Move backwards finding double quotes
         */
        do
        {
            tokenLength++;
            compLength--;
        } while (Comp[compLength] != '"' && compLength);
    }

    /*
     * Add one for the space just before it
     */
    tokenLength++;
    compLength--;
}
/*
 * EVERYTHING ELSE
 * This else handles single word tokens.
 * i.e. not special case, e.g. 'if'
 */
else
{
    /*
     * Count up the chars in the char part
     */
    while (Comp[compLength] != ' ' && Comp[compLength] != '\n' && compLength)
    {
        tokenLength++;
        compLength--;
    }

    /*
     * Count up the chars in the indent part
     */
    while (Comp[compLength-1] == ' ' || Comp[compLength-1] == '\n' && compLength)
    {
        tokenLength++;
        compLength--;
    }
}
return (tokenLength);
}
```

90/06/07
16:44:37

demorgans.c

1

DEMORGANS

Purpose: DeMorgans converts a comp into a form that Troy's CLIPS help routine can deal with. Basically, it just puts double quotes around certain CLIPS key words.

Designer: J. Harold Taylor/SDC

Programmer: J. Harold Taylor/SDC

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Harold Taylor 1-21-87

Added double quotes around '(' and ')'

External Interfaces

```
*****/
#include <ctype.h>

demorgans (the_comp)
char *the_comp;

{
    char temp_comp[1000], end_string[1000];
    int i,j, string_counter, start_of_comp;
    char ch, new_ch, temp[1000], token[300];

    reset_clips ();
    temp_comp[0] = '\0';
    end_string[0] = '\0';

    strcpy (temp_comp, the_comp);

    /* First we need to strip all of the header stuff from the comp */
    start_of_comp = 0;
    string_counter = 0;
    temp[0] = '\0';
    i = 0;
    while ( !start_of_comp)
    {
        ch = temp_comp[i];
        i++;
        if (ch == '\n')
        {
            ch = temp_comp[i];
            /* if we find a newline followed by an 'i' */
            /* then we've found the start of the comp */
            if (ch == 'i')
            {
                start_of_comp = 1;
                /* Build a new string without the header */
                /* stuff in it */
                while (temp_comp[i] != '\0')
                {
                    temp[string_counter] = temp_comp[i];
                    string_counter ++;
                    i++;
                }
                temp[string_counter] = '\0';
            }
        }
    }
    strcpy(temp_comp, temp);
    printf ("\033GC\n");
}
```

```
strcpy(end_string, "");
i=j=0;
while (temp_comp[i] != '\0')
{
    /*****
    Set token equal to the temp_comp until you hit a
    space or line feed.
    *****/
    for (j=0; ((isctrl(temp_comp[i]) == 0) &&
        (isspace(temp_comp[i]) == 0)); j++, i++)
        token[j] = temp_comp[i];
    token[j] = '\0';
    /*****
    Now if the token is one of the special
    tokens that clips needs to know about
    mark it with double quotes.
    *****/
    if (strcmp(token, "=") == 0)
        strcat(end_string, "\042=\042");
    else if (strcmp(token, "(") == 0)
        strcat(end_string, "\042(\042");
    else if (strcmp(token, ")") == 0)
        strcat(end_string, "\042)\042");
    else if (strcmp(token, ">") == 0)
        strcat(end_string, "\042>\042");
    else if (strcmp(token, ">=") == 0)
        strcat(end_string, "\042>=\042");
    else if (strcmp(token, "<") == 0)
        strcat(end_string, "\042<\042");
    else if (strcmp(token, "<=") == 0)
        strcat(end_string, "\042<=\042");
    else if (strcmp(token, "<>") == 0)
        strcat(end_string, "\042<>\042");
    else if (strcmp(token, "+") == 0)
        strcat(end_string, "\042+\042");
    else if (strcmp(token, "-") == 0)
        strcat(end_string, "\042-\042");
    else if (strcmp(token, "/") == 0)
        strcat(end_string, "\042/\042");
    else if (strcmp(token, "**") == 0)
        strcat(end_string, "\042*\042");
    else
        strcat(end_string, token);

    strcat(end_string, " ");
    i++;
}

/* Actually run Clips */
assert (end_string);
run (-1);
}
```

90/06/07
16:27:42

dir.h

1

```
/* @(#)dir.h 20.1 (MASSCOMP) 5/5/87 compiled 3/16/88 */
/*
 * UFS directory entry structure
 */
#ifndef UFS_DIRSIZ
#define UFS_DIRSIZ 14
#endif
struct ufs_direct
{
    ino_t    d_ino;
    char     d_name[UFS_DIRSIZ];
};
```

90/06/07
16:27:44

edit.c

1

```
*****
EDIT
Purpose: Edit allows the user to ved/vi-edit his file within the
        Work Area window.

Designer:  Troy Heindel

Programmer: Troy Heindel

Date: 4/5/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:
-----
```

External Interfaces

```
ask()      -- prompts the user for a char string
inform()   -- display text in the message window
changeFont() -- change text font for selected window
*****/
#include "code.h"
#include "color.h"
#include <termio.h> /* used for setting up the keyboard */

struct termio orig_tty; /* used for setting up the keyboard */
struct termio raw_tty; /* used for setting up the keyboard */

edit ()
{
    char temp[400], /* Text message repository */
        editor[10], /* The name of the editor (ved/vi) */
        highLevelFile[PATH_LEN], /* Path for CODE lang. file */
        compVarsFile[PATH_LEN], /* Path for comp variable file */
        compCFile[PATH_LEN], /* Path for comp translation file */
        groupCFile[PATH_LEN]; /* Path for the group source file */

    /*
     * Allow the user to choose their editor (ved or vi).
     */
    do
    {
        if(ask("Would you like to use the vi or ved editor, or just (Q)uit this?",GREEN,editor)==DEFAULT)
            strcpy (editor, "ved");
        if (editor[0] == 'q' || editor[0] == 'Q')
            return(OK);
    } while (strcmp (editor, "vi") != 0 && strcmp (editor, "ved") != 0);

    clearWA();

    /*
     * Create absolute paths to the high level,
     * variable, comp C, and group C files.
     */
    sprintf (highLevelFile, "%s/%s/%s.h", CodeGroups, GroupName, CompName);
    sprintf (compVarsFile, "%s/%s/%s.v", CodeGroups, GroupName, CompName);
    sprintf (compCFile, "%s/%s/%s.c", CodeGroups, GroupName, CompName);
    sprintf (groupCFile, "%s/%s/%s.c", CodeGroups, GroupName, GroupName);

    /*
     * Let's show our guests know what is
     * behind curtain number A, B, C, D
     */
    inform("Editor buffers> A:high level, B:variables, C:C file, D:group C file",PURPLE,0);

    /*
     * Go to a smaller font in order to avoid the dreaded window
     * wrap and header problems.
     */
    changeFont (WORK_AREA, MEDIUM);

    printf ("\033GE\n"); /* Select work area window */
    printf ("\033Gc\n"); /* Turn text cursor on */
}
```

90/06/07
16:27:44

edit.c

2

```
printf ("\033Gb\n"); /* Select blinking cursor */
printf ("\033[H\n"); /* Place cursor in Home position*/

/*
 * Let the user know they are in a different kind of place by
 * redrawing the window in black; very scary.
 */
mgihue (BLACK);
mgrbox (0.1573, 0.0820, 0.9990, 0.9100);
mgihue (YELLOW);

/*
 * Go to original tty mode for edit inputs.
 */
ioctl (fileno (stdout), TCSETAW, &orig_tty);
ioctl (fileno (stdout), TCFLSH, 0);

/*
 * Make the system call which calls the editor, checking, of
 * of course, for errors.
 */
sprintf (temp, "%s %s %s %s %s 2>>/tmp/code.err", editor, highLevelFile, compVarsFile,
        compCFile, groupCFile);
if (system (temp) == ERROR)
{
    ask("Unable to complete system call - Hit [RETURN] to continue",RED,temp);
}
/*
 * Go back to raw mode.
 * Re-draw the Work Area and Keyboard Input Area in blue,
 * re-draw the Yellow lines that separate the windows,
 * turn the text cursor off, Clear the Window.
 */
ioctl(fileno(stdout), TCSETAW, &raw_tty);
mgihue (BLUE);
mgrbox (0.1573, 0.0010, 0.9990, 0.9100); /* Work Area */
mgrbox (0.1573, 0.0810, 0.9990, 0.1580); /* Keyboard input area */
mgihue (YELLOW);
mgrl (0.1563, 0.0800, 1.0000, 0.0800); /* MESSAGES/PROMPTS Title Line */
mgrl (0.1563, 0.1600, 1.0000, 0.1600); /* KEYBOARD INPUT Title Line */
printf ("\033Gac\n"); /* Turn text cursor off */
printf ("\033[2J\n"); /* Clear the Work Area */
)
```

90/06/07
16:27:53

get_header.c

1

GET_HEADER

Purpose: Get_header gets the header info on a new comp.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/17/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

External Interfaces

cleanSlate() -- Initializes global variables.
displayWA() -- displays a string in the work area window
ask() -- Prompt the user for keyboard input

```
*****/
#include <time.h>
#include <pwd.h>
#include "code.h"

get_header ()
{
    long int timesec,days,hrs,mins,secs; /* used in time calculations */
    int getuid (); /* UNIX routine to get user information */
    char time_str[50]; /* String to build time with */
    struct passwd *getpwuid (); /* UNIX routine to get user information */
    struct passwd *user_pass; /* A structure to put the user info into */
    extern int time(); /* time subroutine */

    /*
     * Initialize variables.
     */
    cleanSlate();

    /*
     * Append the current time to the Comp string.
     */
    strcpy (Comp,"/*****\n");
    timesec = time(0);
    days = timesec/86400;
    hrs = timesec/3600;
    mins = timesec/60;
    secs = timesec - mins*60;
    mins = mins - hrs*60;
    hrs = hrs-days*24;
    hrs = hrs - 5;
    sprintf(time_str,"%03d:%02d:%02d",days,hrs,mins,secs);
    strcat (Comp," Creation Time: ");
    strcat (Comp, time_str);

    /*
     * Append the user's name.
     */
    strcat (Comp, " Author: ");
    user_pass = getpwuid (getuid());
    strcat (Comp,user_pass->pw_gecos);
    strcat (Comp,"\n");

    /*
     * Append the group & comp names.
     */
    strcat (Comp," Group Name: ");
    strcat (Comp, GroupName);
    strcat (Comp,"\t\t Comp Name: ");
    strcat (Comp, CompName);
    strcat (Comp, "\n");

    /*
     * Append the purpose to the Comp string within
```


90/06/07
16:27:53

get_header.c

2

```
    * the global CompInfo structure.
    */
    strcat (Comp, " Purpose: ");
    displayWA(Comp);
    ask("What is the purpose of this computation?", GREEN, CompInfo[CompNumber].purpose);
    strcat (Comp, CompInfo[CompNumber].purpose);
    strcat (Comp, "\n*****");

    /*
     * Set the PrevChoice to COMMENT since the header is just
     * a large comment.
     */
    PrevChoice[ChoiceCounter++] = COMMENT;
    NeedToSave = TRUE;
}
```

90/06/07
16:30:34

get_name.c

1

GET_NAME

Purpose: Get_name displays a formatted list of groups or comps, prompts the user for a number which is associated with with the comp or group and resets the name and number to the newly chosen one.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/20/89

Version: 2.0

Project: CODE (Comp Development Environment)

External Interfaces

inform() -- displays a text message in message window
readCompNames() -- reads comp names from file
clearWA() -- clear the work area window
displayWA() -- writes a string to the work area window
ask() -- prompts the user for a string

```

/*****
#include "code.h"
#include "color.h"
#define Numpagecols 3

get_name(numOfItems, theType, nameToGet, numToGet)
int    numOfItems; /* The number of items to list */
char   *theType;   /* either "Group" or "Comp" */
char   *nameToGet; /* The chosen name is placed here */
int    *numToGet;  /* The chosen number is placed here */
{
    int    i,j,k,          /* Counters used for looping */
    tempNumber,            /* Used in sorting dispositions */
    rc,                   /* The returned value from ask() */
    atoi();               /* Convert string to integer */

    char    message[500], /* for strings displayed to user */
    reply[50],           /* for user input */
    big_str[200],        /* for formatting lines of the list */
    fmtList[6500];       /* for formatting the entire list */

    FILE    *openFile(); /* A handy file opening routine */

    struct group_info_struct nameList[MAX_COMPS]; /* Place to put the names */

    /*
     * Let's leave if there is nothing to list.
     */
    if (numOfItems < 1)
    {
        sprintf (message,"There are not any %s(s) to list", theType);
        inform(message, GREEN, 2);
        return (ERROR);
    }

    if (strcmp (theType, "Comp") == 0)
    {
        /*
         * If there is only one comp, then it is the one
         * and we return with it
         */
        if (numOfItems < 2)
        {
            strcpy (nameToGet, CompInfo[0].name);
            *numToGet = 0;
            return OK;
        }

        /*
         * If there are more than one comps, then put their
         * names into a temp struct for viewing purposes.
         */
    }
}

```

```
for (i=0;i<numOfItems;i++)
{
    strcpy (nameList[i].name, CompInfo[i].name);
    nameList[i].disposition = CompInfo[i].disposition;
}
}
else if (strcmp (theType, "Group") == 0)
{
    /*
     * If there is only one group, then it is the one
     */
    if(numOfItems<2)
    {
        /*
         * Read in the comps associated with this group.
         */
        strcpy (nameToGet, GroupInfo[0].name);
        *numToGet = 0;
        readCompNames();
        return OK;
    }

    /*
     * If there are more than one groups, then put their
     * names into a temp struct for viewing purposes.
     */
    for (i=0;i<numOfItems;i++)
    {
        nameList[i] = GroupInfo[i];
    }
}
else
{
    /*
     * This case should never happen.
     */
    sprintf (message,"get_name: Recieved invalid argument '%s'",theType);
    inform(message,RED,4);
    return (ERROR);
}

/*
 * Display the names in the Work Area
 */
fmtList[0] = 0; /* clean of the string */
sprintf (fmtList, "\t\t\t\tListing of Available %ss\n\n", theType);
sprintf (message,"%#    %s Name Diposition #    %s Name Diposition #    %s Name Diposition\n", theType,theTy
pe,theType);
strcat (fmtList, message);
strcat (fmtList,"==== ===== \n");
for (j=1;j<=numOfItems;j++)
{
    if (j % Numpagecols)
    {
        sprintf (big_str, "%d ", j);          /* Build group number */
        if (j<10) strcat (big_str, " ");      /* Justify one digit numbers */
        strcat (big_str, nameList[j-1].name); /* The group name */
        for (k=strlen(big_str);k<16;k++)      /* Add some filler spaces */
            strcat (big_str, " ");
        if (nameList[j-1].disposition == INCOMPLETE)
            sprintf (message, "INCOMPLETE");
        else if (nameList[j-1].disposition == COMPLETE)
            sprintf (message, "COMPLETE ");
        else if (nameList[j-1].disposition == ERROR)
            sprintf (message, "ERROR ");
        else sprintf (message, "INSTALLED ");
        strcat (big_str, message);
        strcat (fmtList, big_str);
        for (k=strlen(big_str);k<28;k++)
            strcat (fmtList, " ");
    }
    else
    {
        sprintf (big_str, "%d ", j);
        if (j<10) strcat (big_str, " ");
        strcat (big_str, nameList[j-1].name);
        for (k=strlen(big_str);k<16;k++)
            strcat (big_str, " ");
    }
}
```

```
        if (nameList[j-1].disposition == INCOMPLETE)
            sprintf (message, "INCOMPLETE\n");
        else if (nameList[j-1].disposition == COMPLETE)
            sprintf (message, "COMPLETE\n");
        else if (nameList[j-1].disposition == ERROR)
            sprintf (message, "ERROR\n");
        else sprintf (message, "INSTALLED\n");
        strcat (big_str, message);
        strcat (fmtList, big_str);
    }
}
clearWA();
displayWA(fmtList);

/*
 * Get the number of the name from the user, until
 * they get it right.
 */
do
(
    sprintf(message, "Please type the %s number, or (Q)uit this", theType);
    rc = ask(message, GREEN, reply);

    /*
     * If they want to quit, clean up and return
     */
    if (reply[0] == 'q' || reply[0] == 'Q')
    {
        clearWA();
        return(ERROR);
    }

    /*
     * If the just hit return, then give them the
     * first comp or group in the list, otherwise
     * atoi() their reply to get chosen number.
     */
    if(rc==DEFAULT)
        *numToGet = 0; /* Default to the first one */
    else *numToGet = atoi (reply) - 1;

    /*
     * Check to see that the number corresponds to a name.
     */
    if (*numToGet < 0 || *numToGet >= numOfItems)
    {
        sprintf(message, "The number %s does not belong to any %s", reply, theType);
        inform(message, PURPLE, 2);
        rc = ERROR;
    }

    /*
     * Now that we know that it is a good number let's get
     * a name with the number.
     */
    else strcpy (nameToGet, nameList[*numToGet].name);

) while (rc == ERROR);

clearWA();

/*
 * If theType is Group then read in the comps associated
 * with the group.
 */
if(strcmp (theType, "Group") == 0)
    readCompNames();

return (OK);
}
```

90/06/07
16:30:38

getwd.c

1

```
static char SccsId[] = "@(#)getwd.c 2.2 8/17/88 08:51:56";
/*-----
 *
 * Function:      getwd
 *
 * Entry specification:
 *   char *getwd(work_area)
 *   char *work_area;
 *
 *
 * Description:
 *   This function finds the current directory, it is based on the UCB getwd()
 *   routine. It was written to temporarily replace the same function in the
 *   Masscomp C library (which had severe problems on certain machines).
 *
 *
 * Inputs:
 *   work_area      Pointer-> work space to build path
 *
 * Returns:      Pointer-> work area with completed path
 *
 * External references:
 *   stat() fstat()
 *   bcopy()
 *   open() read() write() close()
 *   chdir()
 *   strins() strlen()
 *   exit()
 *
 * Resources used:  None
 * Limitations:    None
 * Assumptions:    None
 *
 * Written by:  Tom Silva, Bruce G. Jackson & Associates
 *
 * Traceability:
 *   Version  Date      Description
 *   -----
 *   1.0      07/01/87  initial version
 *   1.1      08/01/88  modified for compatabilty with RTU 4.0  AA
 *
 * Notes:      None
 *-----*/

/*-----
 * Includes.
 */
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "dir.h" /* AA */
#define direct ufs_direct /* AA */
#define DIRSIZ UFS_DIRSIZ /* AA */

/*-----
 * Local definitions of variables, structures, and compiler directives
 * which are "owned" by this file.
 */
union safe_dir
{
    struct direct entry;
    char filler[sizeof(struct direct) + 4];
};

/*-----
 *
char *getwd(work_area)
char *work_area;
{
    struct stat current, next;
    int root_device, root_inode;

    /*-----
 * Get the status of the root directory in order to save it's device and
 * inode. Initialize the caller's string work area to an empty string.
 */

```

```
*/
stat("/", &current);
root_device = current.st_dev;
root_inode = current.st_ino;
work_area[0] = 0;

/*-----
 * Get the status of the current directory.
 * Loop until the current directory is the root directory.
 * Copy the status of the next directory to the current after every loop.
 */
for (stat(".", &current);
     current.st_dev != root_device || current.st_ino != root_inode;
     bcopy(&next, &current, sizeof(current)))
{
    int nextdir, same_disk;
    union safe_dir dir;

    /*-----
     * Open then next directory for reading, status it, then change to it.
     * We still regard 'current' as the current directory under test even
     * though it's not the true current directory at this time.
     * Figure out whether the next directory is on the same disk device as
     * the current directory. This info is needed inside the loop.
     */
    nextdir = open("../", O_RDONLY);
    if (nextdir < 0) bad_news("getwd: Can't open ../\n");
    fstat(nextdir, &next);
    chdir("../");
    same_disk = (current.st_dev == next.st_dev);

    /*-----
     * Loop reading from the next directory, try to find the current
     * directory.
     */
    for (;;)
    {
        /*-----
         * Read one directory entry. Put it into a 'safe' structure. The
         * safe structure will provide padding on the end just in case we
         * run into a 14 character filename.
         */
        if (read(nextdir, &dir.entry, sizeof(struct direct)) < sizeof(struct direct))
            bad_news("getwd: Can't read ../\n");

        /*-----
         * If the two directories are from the same disk device then all
         * we do is check the contents of the next directory for the inode
         * of the current directory. Once we've found it then we have a
         * name.
         */
        if (same_disk)
        {
            if (dir.entry.d_ino == current.st_ino) break;
        }

        /*-----
         * If the two are from different disks then the inodes in the next
         * directory have no relation to the current directory's inode.
         * In this case we must check each of the files in the next
         * directory (which is really the current working directory).
         * We do a status of each file by name (after correcting for any
         * 14 character names). If the tested file has the same inode AND
         * is on the same device as the current directory then we've found
         * the right name.
         */
        else
        {
            struct stat match;

            dir.entry.d_name[DIRSIZ] = 0;
            stat(dir.entry.d_name, &match);
            if (match.st_ino == current.st_ino && match.st_dev == current.st_dev) break;
        }
    }

    /*-----
     * Close the directory file. Adjust the current directory's name in

```

```

* case it's 14 characters, and prepend a slash.
* The slash actually ends up in that part of the 'direct' structure
* that holds the inode. This coercion works as long as that structure
* has the inode element first.
* Insert the new name at the beginning of the caller's work string.
* The inserted name actually starts at the slash.
*/
close(nextdir);
dir.entry.d_name[-1] = '/';
dir.entry.d_name[DIRSIZ] = 0;
strins(work_area, dir.entry.d_name - 1);
}

/*-----
* If there isn't anything in the caller's string by now then we're at the
* root directory, so make the correct string.
*/
if (!work_area[0])
{
work_area[0] = '/';
work_area[1] = 0;
}

/*-----
* Change directory backwards to where we started, return the address of
* the work area to the user.
*/
if (chdir(work_area)) bad_news("getwd: Can't chdir back\n");
return work_area;
}

/*-----
*-----*/
static bad_news(msg)
char *msg;
{
write(2, msg, strlen(msg));
exit(1);
}

```

90/06/07
15:30:44

init_code.c

1

/******

INIT_CODE

Purpose: To initialize all flags and strings at start of CODE

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/14/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by: Terri Murphy

External Interfaces

inform() -- display text in the message window

#include "code.h"

#include <termio.h> /* used for setting up the keyboard */

struct termio orig_tty; /* used for setting up the keyboard */
struct termio raw_tty; /* used for setting up the keyboard */

init_code ()

{
 char knowledge_base[60], /* String to hold CLIPS KB name */
 message[200], /* char string for display */
 reply[50], /* String for user input */
 getenv(), / Used to obtain paths */
 listUserFuncs[250], /* String for storing/listing user functions */
 AM[PATH_LEN], /* path for algo manager */
 ch1, /* ch1 & ch2 are used for trashing the */
 ch2; /* header on the tag_msid_tbl file */

int i,j, /* Counters */
 rc; /* A return code */

FILE *ptr, /* Pointer */
 openFile(); / A handy file opening routine */

/*
 * Remove the old system errors caused while running CODE.
 */

if(system("rm /tmp/code.err 2>> /tmp/code.err")!=OK)
 inform("Unable to remove /tmp/code.err",PURPLE,2);

/*
 * raw input set-up
 *
 * Get the parameters associated with the terminal
 * and store them in both orig and raw.
 * Modify the parameters associated with the raw mode.
 */

ioctl (fileno (stdout), TCGETA, &orig_tty);
ioctl (fileno (stdout), TCGETA, &raw_tty);
raw_tty.c_lflag &= ~(ICANON | ECHO | ISIG);
raw_tty.c_iflag &= ~(ISTRIP | IXON | IXANY | INLCR | ICRNL | IGNCR);
raw_tty.c_cc[VMIN] = 1;
raw_tty.c_cc[VTIME] = 0;

ioctl(fileno(stdout), TCSETAW, &raw_tty); /* Begin in raw tty mode. */

/*
 * Load knowledge base for CLIPS
 */

/* inform("Initializing CLIPS Rule Base...", PURPLE, 2);
strcpy (knowledge_base, CODE);
strcat (knowledge_base,"/DeMorgan.clp");
init_clips ();
load_rules (knowledge_base);
*/

/*
 * Set up to paths which are used in this and other modules

2

```

*/
rc = 0;
strcpy (RTDS, getenv("RTDS"));
if (RTDS[0] == 0)
{
    ask("An environment variable was not found for 'RTDS'\nPlease setup your login file with this variable.
-- Hit [RETURN]", RED, reply);
    return ERROR;
}

sprintf (SignalTbl, "%s/tables/signal_table", RTDS);      /* file */
sprintf (MSIDTbl, "%s/tables/tag_msid_tbl", RTDS);       /* file */
sprintf (CodeGroups, "%s/rtds/code/groups", RTDS);       /* directory */
sprintf (GroupNamesFile, "%s/GroupNames", CodeGroups);   /* file */
sprintf (CodeDocs, "%s/rtds/code/docs/code_doc", RTDS); /* file */
sprintf (Code, "%s/rtds/code", RTDS);                   /* directory */
sprintf (UserFuncsLib, "%s//rtds/lib/libuserfuncs.a", RTDS); /* library */
sprintf (AM, "%s/rtds/am", RTDS);                       /* directory */
sprintf (AMSupport, "%s/rtds/am/support", RTDS);         /* directory */
sprintf (AMGroups, "%s/rtds/am/groups", RTDS);           /* directory */

/*
 * Let's check the accessibility of some paths to see if they exist
 * and create them if they do not, and exit if we can not create them.
 */
rc += CheckOrMkdir(Code);
rc += CheckOrMkdir(CodeGroups);
rc += CheckOrMkdir(AM);
rc += CheckOrMkdir(AMSupport);
rc += CheckOrMkdir(AMGroups);

if(rc) cleanExit(); /* Exit if we had difficulty */

/*
 * Load the msid list and info for msid validation in put_msid.c
 */
if(ptr = openFile (MSIDTbl, "r", "init_code"))
{
    inform("Loading the MSID list into memory...", PURPLE, 0);
    MSIDCount = 0;
    /*
     * Rip the header off the file
     */
    while (( (ch1 = fgetc (ptr)) != EOF) && (!(ch1 == '/' && ch2 == '*')))
    {
        ch2 = ch1;
    }
    /*
     * Loop through the file filling the global structure MSIDTable
     * with all those lovely msid's and related info.
     */
    while (fscanf (ptr, "%s %c %c %c %c %c %s %c %c [^\n]", MSIDTable[MSIDCount].name,
        MSIDTable[MSIDCount].type,
        MSIDTable[MSIDCount].nomenclature) != EOF)
    {
        /*
         * Convert the one character msid type definition
         * to a full word type definition which is the
         * standard for CODE.
         */
        if (MSIDTable[MSIDCount].type[0] == 's')
            strcpy (MSIDTable[MSIDCount].type, "short");
        else if (MSIDTable[MSIDCount].type[0] == 'i')
            strcpy (MSIDTable[MSIDCount].type, "int");
        else if (MSIDTable[MSIDCount].type[0] == 'S')
            strcpy (MSIDTable[MSIDCount].type, "char");
        else if (MSIDTable[MSIDCount].type[0] == 'f')
            strcpy (MSIDTable[MSIDCount].type, "float");
        else if (MSIDTable[MSIDCount].type[0] == 'd')
            strcpy (MSIDTable[MSIDCount].type, "double");
        /*
         * Increment the msid record counter
         */
        MSIDCount++;
    }
    fclose (ptr);
}

```

```
/*
 * Create and execute the system command which
 * will create a file listing of user functions.
 */
sprintf (listUserFuncs, "ar t %s > /tmp/user_funcs 2>>/tmp/code.err", UserFuncsLib);
system(listUserFuncs); /* If we can't do this, who cares! */

/*
 * Now create a path for the file to be read
 */
strcpy (listUserFuncs, "/tmp/user_funcs");

/*
 * Open the newly created file /tmp/user_funcs
 * and read out the names of the user functions.
 */
if (!(ptr = fopen (listUserFuncs, "r")))
{
    inform("tmp/user_funcs file was not found",PURPLE,2);
}
else
{
    NumberOfUserFuncs = 0; /* start at 0 */
    /*
     * Loop through the file filling the global structure MSIDTable
     * with all those lovely msid's and related info.
     */
    fscanf (ptr, "%*[^\\n]"); /* Rip the header */
    while (fscanf (ptr, "%s", UserFuncs[NumberOfUserFuncs]) != EOF)
    {
        UserFuncs[NumberOfUserFuncs][strlen(UserFuncs[NumberOfUserFuncs])-2]='\0';
        /*
         * Make sure the function is not a CODE reserved word.
         */
        if (strcmp ("if",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("then",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("and",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("or",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("not",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print1",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print2",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print3",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print4",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print5",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("set",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp (">",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp (">=",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("<",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("<=",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("=",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("else",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("endif",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("8",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("(",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp (")",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("+",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("-",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("**",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("/",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("bitXor",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("exp",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("log",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("cos",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("acos",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("sin",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("asin",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("tan",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("atan",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("sqrt",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("power",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("bitAnd",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("bitOr",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("shiftL",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("shiftR",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("p",UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("",UserFuncs[NumberOfUserFuncs]) == 0)
        {
            NumberOfUserFuncs++;
        }
    }
}
```

```
    sprintf (message, "'%s' is a CODE reserved word and therefore can not be used as a user function.
\nIgnoring %s", UserFuncs[NumberOfUserFuncs], UserFuncs[NumberOfUserFuncs]);
    inform(message,RED,2);
}
/*
 * Increment the function counter.
 */
else NumberOfUserFuncs++;
}

/*
 * Let's not forget to clean up after ourselves.
 */
if(system ("rm /tmp/user_funcs 2>>/tmp/code.err")!=OK)
    inform("Unable to remove /tmp/code.err",PURPLE,2);

/*
 * Read the GroupNames file into memory. If we have a problem
 * we exit since no useful work can be done without reading
 * and writing this file.
 */
if (readGroupNames() == ERROR)
    cleanExit();
}

/*****
CheckOrMkdir(path)
char path[]; /* path to be checked or made */
{
    char message[250]; /* for displaying a user message */

    if(access(path,F_OK)==ERROR)
    {
        /*
         * Try to create the directory.
         */
        sprintf(message,"Creating the '%s' directory...",path);
        inform(message,PURPLE,1);
        if(mkdir(path,511)!=OK)
        {
            sprintf(message,"Could not create directory '%s'. Check code installation instructions\nfor informat
ion to correct this. Hit [RETURN] to exit code.",GREEN,path);
            ask(message,PURPLE,message);
            return ERROR;
        }
    }
    return OK;
}
```

90/06/07
16:33:27

init_gp.c

1

```
/*  
initGraphics
```

Purpose: Initialize the windows and graphics.

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 12/13/88

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
color_valid()  -- redraws the tokens in red or yellow  
background()  -- draws the background stuff  
hello_screen() -- draws the credits
```

```
*****/  
#include "code.h"  
#include "cursor.h"  
#include <pwd.h>
```

```
initGraphics ()
(
    int      resolution[5],      /* Used to place the screen resolution in; winhardget() */
    getuid ();                  /* For getting the name of the user */

    struct passwd *getpwuid ();
    struct passwd *user_pass;    /* A structure to put the user info into */

    char userName[15];          /* The user's name is set in get_header */

    /* ignore ^C
    */
    /* signal (SIGINT, SIG_IGN); */

    mgiasngp(0,0);              /* Assign the graphics processor */

    /*
    * Get the color map so we can restore it before exiting
    */
    mgigetcms (0, 32, ColorMap);

    /*
    * Assign colors to color map
    */
    mgicm (1,mgfcns("dark vivid blue"));
    mgicm (2,mgfcns("red"));
    mgicm (4,mgfcns("orange"));
    mgicm (8,mgfcns("yellow"));
    mgicm (12,mgfcns("purple"));
    mgicm (16,mgfcns("light vivid green"));
    mgicm (18,mgfcns("white"));

    /*

    -----
    DEFAULT COLORS
    -----
    dark vivid blue
    red
    orange
    yellow
    purple
    light vivid green
    white

    */

    /*
    * Create the text windows and associate them with the graphics windows.
    * 1 - reserved, 2 - reserved, 3 - Work Area, 4 - Prompt, 5 - Edit,
    * 7 - Messages.
    */
    mgidefw (3);                /* This is the Working Area window */
    mgrpw (3, 0, 0.1563, 0.1600, 1.0000, 0.9110);
    mgitw gw (3,3);             /* This associates a text window */
                                /* with the working area window */

    mgidefw (4);                /* This is the Keyboard Input window */
    mgrpw (4, 0, 0.1563, 0.0800, 1.0000, 0.1600);
    mgitw gw (4,4);             /* This associates a text window */
                                /* with the keybd input window */

    mgidefw (5);                /* This is the Edit window */
    mgrpw (5, 0, 0.1573, 0.0800, 0.9990, 0.9100);
    mgitw gw (5,5);             /* This associates a text window */
                                /* with the working area window */

    mgidefw (7);                /* This is the Message/Prompts window */
    mgrpw (7, 0, 0.1563, 0.0000, 1.0000, 0.0800);
    mgitw gw (7,7);             /* This associates a text window */
                                /* with the Message/Prompts window */

    /*
    * Assume the large font
    */
    mgifetchgf (Font, "9x11");
    mgifetchgf (BoldFont, "9x11_b");
    mgifetchgf (ItalicFont, "9x11_i");
```

```
mgifetchgf (HugeFont, "10x11");

/*
 * Get the horizontal and vertical screen resolution,
 * and set global variables to these values.
 */
winhardget (resolution, 3);

if (resolution[0]-1 > 1000)
{
    mgitf (3, LargeTextFont, -1, 1); /* win, big font, subscripts, wrap */
    mgitf (4, LargeTextFont, -1, 1); /* win, big font, subscripts, wrap */
    mgitf (5, LargeTextFont, -1, 1); /* win, big font, subscripts, wrap */
    mgitf (7, LargeTextFont, -1, 1); /* win, big font, subscripts, wrap */
}
else
{
    mgitf (3, MediumTextFont, -1, 1); /* win, little font, subscripts, wrap */
    mgitf (4, MediumTextFont, -1, 1); /* win, little font, subscripts, wrap */
    mgitf (5, MediumTextFont, -1, 1); /* win, little font, subscripts, wrap */
    mgitf (7, MediumTextFont, -1, 1); /* win, little font, subscripts, wrap */
}

/*
 * Cursor definition and loading
 */
printf ("\033Gac\n"); /* Turn off the text cursor */
mgiclearpln (2,-1,0); /* Clear all planes */
mgigf (ItalicFont); /* Write the background in bold font */
background (); /* Draw the graphics background */
mgigf (Font); /* Do all else in normal font */
color_valid (PrevChoice[ChoiceCounter - 1], 99);

/*
 * Draw the hello screen if the user is not one of the developers!
 */
user_pass = getpwuid (getuid());
strcpy (userName, user_pass->pw_name);
if ((strcmp (userName, "troy", 4) != 0) &&
    (strcmp (userName, "terri", 5) != 0))
    hello_screen (); /* Draws the CODE HELLO screen */
mgihue (GREEN);
mgiloadcurs (7,7,4,Shuttle);
mgicursmode (15);
mgrcursxy (2, 6, 0.1923, 0.9283); /* Position the cursor */
}
```

/*****

HELLO

Purpose: Hello draws the hello window with credits.

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 12/1/88 (my birthday!)

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

*****/
#include "code.h"

hello_screen ()

```
{
    float rx[5], ry[5]; /* for drawing the thick lines */

    mgifb (1, 2);      /* Show 1, modify 2 */

    mgifetchvf (5, "compro");
    mgrvfcontrol (2., 12., 1);
    mgrvfs (.1746, .8262, 0, "CODE");

    mgrvfcontrol (1.2, 1., 0);
    mgrvfs (.18, .4500, 0, "Computation");
    mgrvfs (.18, .4000, 0, "Development");
    mgrvfs (.18, .3500, 0, "Environment");

    mgrvfcontrol (0.8, 1., 0);
    mgrvfs (.5000, .8500, 0, "A natural language tool for");
    mgrvfs (.5000, .8000, 0, "building real-time algorithms.");
    mgrvfs (.5000, .5900, 0, "Version 2.0b   Nov. 23, 1988");

    mgigf (HugeFont);
    mgrgfs (.5000, .5000, 0, "Designed & Built by:");
    mgrgfs (.5000, .4500, 0, "Troy A. Heindel");
    mgrgfs (.5750, .4250, 0, "&");
    mgrgfs (.5000, .4000, 0, "Terri B. Murphy");
    mgrgfs (.5000, .3500, 0, "National Aeronautics & Space Administration");
    mgrgfs (.5000, .3250, 0, "Johnson Space Center");
    mgrgfs (.5000, .3000, 0, "Mission Operations Directorate");
    mgrgfs (.4200, .1800, 0, "(Click anywhere to begin!)");

    mgigf (Font);

    rx[0] = .1800;
    rx[1] = .424;
    rx[2] = .424;
    rx[3] = .1800;
    rx[4] = .1800;
    ry[0] = .8713;
    ry[1] = .8713;
    ry[2] = .578;
    ry[3] = .578;
    ry[4] = .8713;
    mgiwidth (5, 0);
    mgrtls (5, rx, ry);

    /* C */
    mgrbox (.2000, .7600, .2250, .7700);
    mgrfc (.2125, .7100, .0125);
    mgrbox (.2000, .6500, .2250, .6600);
    mgrfc (.2125, .6100, .0125);

    /* O */
```

90/06/07
16:33:27

init_gp.c

5

```
mgrbox (.2585, .7600, .2835, .7700);  
mgrbox (.2585, .7050, .2835, .7150);  
mgrbox (.2585, .6500, .2835, .6600);
```

```
/* D */  
mgrbox (.3170, .7600, .3420, .7700);  
mgrfc (.3295, .7100, .0125);  
mgrfc (.3295, .6550, .0125);
```

```
/* E */  
mgrfc (.3880, .7650, .0125);
```

```
/*  
 * Display the hello screen by showing frame buffer 2  
 */  
mgifb (2, 1);  
}
```


90/06/07
16:34:13

install.c

1

INSTALL

Purpose: This routine does all the necessary stuff to move a comp built using CODE into a group of comps in the INCO Algorithm Manager.

Designer: Terri Murphy & Troy Heindel

Programmer: Terri Murphy & Troy Heindel

Date: 6/1/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
openFile() -- generic file opening routine.
save_CompInfo() -- save new comp dispositions.
inform() -- display text in the message window
ask() -- inform(), then prompt for reply
clearWA() -- clears work area window
displayWA() -- display text string in work area window
read_vars() -- read variables for a "complete" comp
variable_exist() -- see if the variable is already in the group vars list
write_comp() -- appends "complete" comps .c to group.c
config_mgmt() -- verify proper type of signals and msids
compile_group -- Compiles the group.c file
*****/
#include "code.h"

struct termio orig_tty; /* used for setting up the keyboard */
struct termio raw_tty; /* used for setting up the keyboard */

FILE *group_file, /* Pointer to the file that the <GroupName>.c code will go in */
*report, /* Pointer to a file for standard errors */
*openFile(); /* A handy file opening routine */

int num_group_msids, /* The number of msids that belong to this group */
numSigsTBDef; /* the number of signals that need to be added to the signal table */

struct sig_tbl_struct sigsTBDef[MAX_NUM_SIGNALS]; /* The array of signals that need to be added
to the signal table */
```

```
install()
{
    FILE      *ptr1,          /* File pointer to the complete group report */
              *sig_tbl_ptr;   /* File pointer to the signal table */

    int        i,             /* Simple index */
              status,          /* return code for function calls */
              installed[MAX_GROUPS], /* Contains the index's of all comps that we
                                      are attempting to install */
              comps_installed, /* The number of comps we are attempting to install */
              group_file_name[PATH_LEN], /* Full path name of the group file */
              comp_var_count, /* The number of variable that belong to the comp that is currently
                                      being processed (Passed to translate). */
              numCompsError, /* The number of comps with disposition == ERROR */
              numCompsComplete, /* The number of comps with disposition == COMPLETE */
              numCompsIncomplete, /* The number of comps with disposition == INCOMPLETE */
              numCompsInstalled, /* The number of comps with disposition == INSTALLED */
              reportError, /* Flag if there's an error in creating the report */
              compileStatus; /* The status returned from compile */

    char        message[180], /* Holds messages, used with ask(),inform() */
              reply[50], /* place to store user responses */
              groupExe[80], /* The name of the group executable file */
              groupRpt[80]; /* The name of the group report */

    struct var_struct this_comps_vars[MAX_COMP_VARS]; /* The array of variables for the comp that
                                                         is currently being processed. (Passed
                                                         to translate. */

    /*
     * If there's a group executable out there, remove it and
     * change the groups disposition to complete.
     */
    sprintf(groupExe, "%s/%s", AMGroups, GroupName);
    sprintf(message, "rm -f %s 2>> /tmp/code.err", groupExe);
    if(system (message)!=OK)
        inform("Unable to remove old group executable", PURPLE, 1);
    GroupInfo[GroupNumber].disposition = COMPLETE;

    /*
     * Initially there are no group msids, installed comps
     * or signals that need defining.
     */
    num_group_msids = comps_installed = 0;
    numCompsInstalled = numCompsComplete = numCompsIncomplete = numCompsError = 0;
    numSigstBDef = 0;

    /*
     * Open a file for installation report.
     */
    sprintf (group_file_name, "%s/%s/install.rpt", CodeGroups, GroupName);
    if (!(report = openFile(group_file_name, "w", "install")))
        return (ERROR);
    fprintf(report, "\nInstallation report for %s:\n\n", GroupName);

    /*
     * We always have at least one group_var "QUALITY".
     */
    NumGroupVars = 1;
    strcpy(GroupVars[0].name, "QUALITY");
    strcpy(GroupVars[0].type, "short");
    strcpy(GroupVars[0].class, "msid");
    GroupVars[0].occurrence = 0;
    GroupVars[0].put_or_get = 0;
    GroupVars[0].lo1_limit = 0;
    GroupVars[0].lo2_limit = 0;
    GroupVars[0].hi1_limit = 0;
    GroupVars[0].hi2_limit = 0;
    strcpy(GroupVars[0].nomenclature, "/* QUALITY */");

    for(i=0; i<NumOfComps; i++)
    {
        /*
         * Get variables for comps and translate comps that
         * satisfy either of the following conditions:
         * 1. All disposition INSTALLED
         */
    }
}
```

```
* 2. All disposition COMPLETE.
*/
if (CompInfo[i].disposition == INSTALLED || CompInfo[i].disposition == COMPLETE)
{
    sprintf(message, "Installing comp : %s", CompInfo[i].name);
    inform(message, PURPLE, 0);
    fprintf(report, "\n%s:\n", CompInfo[i].name);
    status = read_vars(CompInfo[i].name, this_comps_vars, &comp_var_count);
    if (status == ERROR)
    {
        fprintf(report, "\tVariable File Error - Notify Developers\n");
        CompInfo[i].disposition = ERROR;
    }
    status = translate(CompInfo[i].name, this_comps_vars, comp_var_count);
    if (status == ERROR)
    {
        fprintf (report, "\tTranslation Error - Notify Developers\n");
        CompInfo[i].disposition = ERROR;
    }
    else
    {
        fprintf (report, "\tTranslation Complete\n");
        installed[comps_installed++] = i;
    }
}
}

/*
 * If there are no comps_installed don't bother building the
 * <group>.c
 */
if (comps_installed > 0)
{
    /*
     * Let's open a file for the <GroupName>.c code and put
     * the header information in it.
     */
    sprintf (group_file_name, "%s/%s/%s.c", CodeGroups, GroupName, GroupName);
    if (!(group_file = fopen (group_file_name, "w", "install")))
    {
        fprintf(report, "Install: Unable to open %s.\n", group_file);
        fclose(report);
        return (ERROR);
    }

    /*
     * Write our '#include's.
     */
    fprintf(group_file, "#include <rti.h>\n");
    fprintf(group_file, "#include <math.h>\n");
    fprintf(group_file, "#include <stdio.h>\n");
    fprintf(group_file, "#include <values.h>\n");
    fprintf(group_file, "#include <nf.h>\n\n");

    /*
     * Write our '#define's.
     */
    fprintf(group_file, "#define MSID_CNT %d\n", num_group_msids+1);
    fprintf(group_file, "#define COMP_CNT %d\n", comps_installed);
    fprintf(group_file, "#define GROUP_DAT %042s/%s.dat\042\n", AMSupport, GroupName);
    fprintf(group_file, "#define MAX_SIGNAL_LENGTH 80\n");
    fprintf(group_file, "#define MAX_NAME_LEN 11\n\n");

    /*
     * Define our global variables.
     */
    fprintf(group_file, "int *ind_ptr;\n\nint *rate_ptr;\n\nint *nf_ptr;\n\nint nf_index;\n");
    fprintf(group_file, "int iteration_ctr = 0;\n\nint nf[COMP_CNT];\n\n");
    fprintf(group_file, "short value[MSID_CNT];\n\nchar status[MSID_CNT];\n\n");

    /*
     * Loop through the group variable list.
     */
    fprintf(group_file, "/* Signal List */\n");
    for (i=0; i<NumGroupVars; i++)
    {
        /*
         * If we find a signal then define it.
         */
    }
}
```

[illegible]

```

/*
 * Now for the loop....
 */
fprintf(group_file, "\twhile(++iteration_ctr)\n\t{\n");
fprintf(group_file, "\t\tdo\n\t\t{\n\t\t\tgetmsid_off_data_1st(indx_array, value, status);\n\t\t} while (
value[0] == 0);\n\n");
fprintf(group_file, "\t\tind_ptr = ind_array;\n");
fprintf(group_file, "\t\ttrate_ptr = rate_array;\n");
fprintf(group_file, "\t\ttnf_index = 0;\n\n");

for(i=0;i<comps_installed;i++)
{
    fprintf(group_file, "\t\tif((*ind_ptr++ == 1) && {iteration_ctr %% *rate_ptr} == 0)\n");
    fprintf(group_file, "\t\t\t%s();\n", CompInfo[installed[i]].name);
    fprintf(group_file, "\t\t\t++nf_index;\n\t\t\t++rate_ptr;\n\n");
}

fprintf(group_file, "\t}\n\n\n");


/*****
Here's where we write the comp's C code that's
generated by translate.
*****/
for(i=0;i<comps_installed;i++)
{
    if(write_comp(CompInfo[installed[i]].name) == ERROR)
    {
        fprintf(report, "Install: Unable to write %s's 'C' version.\n", CompInfo[installed[i]].name);
        fprintf(group_file, "%s()\n(\n\treturn;\n);\n\n", CompInfo[installed[i]].name);
        CompInfo[installed[i]].disposition = ERROR;
    }
    else CompInfo[installed[i]].disposition = INSTALLED;
}

fclose(group_file);


/*****
Compile and save the group variables and the updated dispositions.
*****/
compileStatus = compile_group();

if(compileStatus == ERROR)
{
    /*
     * If we couldn't compile - then we need to change the comp dispositions
     * back to COMPLETE, and the group disposition to ERROR.
     */
    for (i=0;i<NumOfComps;i++)
    {
        if(CompInfo[i].disposition == INSTALLED)
            CompInfo[i].disposition = COMPLETE;
    }
}

/*****
Report on the new disposition of the comps.
*****/
fprintf(report, "\n\nComp Name\tDisposition\n");
fprintf(report, "_____\n");
for(i=0;i<NumOfComps;i++)
{
    switch(CompInfo[i].disposition)
    {
        case ERROR: fprintf(report,"%s\tERROR\n", CompInfo[i].name);
                    numCompsError++;
                    break;

        case COMPLETE: fprintf(report,"%s\tCOMPLETE\n",CompInfo[i].name);
                        numCompsComplete++;
                        break;

        case INCOMPLETE:fprintf(report,"%s\tINCOMPLETE\n", CompInfo[i].name);
                            numCompsIncomplete++;
                            break;
    }
}

```

```

        case INSTALLED: fprintf(report, "%s\t\tINSTALLED\n", CompInfo[i].name);
                        numCompsInstalled++;
                        break;

        default:      break;
    }
}

fprintf(report, "\n\nTotal Comps:\t\t\t%d\n", NumOfComps);
fprintf(report, "Installed comps:\t\t\t%d\n", numCompsInstalled);
fprintf(report, "Incomplete comps:\t\t\t%d\n", numCompsIncomplete);
fprintf(report, "Comps containing errors:\t\t\t%d\n\n", numCompsError);

if (comps_installed > 0)
{
    if (compileStatus != ERROR)
    {
        /*
         * Since the group was successfully installed, we need to add
         * the signals to the signal table and report what signals were
         * added. But before we do that let's make sure the signal
         * table exists and don't do anything if it does not.
         */
        if (access(SignalTbl, F_OK) != ERROR)
        {
            fprintf(report, "Signals defined in %s as a result of installing group %s:\n", SignalTbl, GroupName);
            if (!(sig_tbl_ptr = fopen (SignalTbl, "a")))
            {
                sprintf(message, "Warning: Could not open '%s' for writing.", SignalTbl);
                inform(message, GREEN, 2);
            }
        }
        else
        {
            /*
             * If we're adding signals to the signal table, we need
             * to need to make sure that we start on a new line.
             */
            if (numSigsTBDef > 0)
                fprintf(report, "\n");
            /*
             * Add the new signals.
             */
            for (i=0; i<numSigsTBDef; i++)
            {
                /*
                 * Since we're adding it to the file - we better also add to
                 * our signal table structure in memory in case we install
                 * again without retrieving!!!
                 */
                strcpy(SignalTable[SignalCount].name, sigsTBDef[i].name);
                strcpy(SignalTable[SignalCount].type, sigsTBDef[i].type);
                strcpy(SignalTable[SignalCount].nomenclature, sigsTBDef[i].nomenclature);
                SignalCount++;
            }
            /*
             * If the type is 'c' the signal table expects
             * the type to be 'S'
             */
            if (sigsTBDef[i].type[0] == 'c')
                sigsTBDef[i].type[0] = 'S';
            fprintf(sig_tbl_ptr, "%s %c %s\n", sigsTBDef[i].name, sigsTBDef[i].type[0], sigsTBDef[i].nomenclature);
            fprintf(report, "\t%s %c %s\n", sigsTBDef[i].name, sigsTBDef[i].type[0], sigsTBDef[i].nomenclature);
        }
        fclose (sig_tbl_ptr);
    }
}

fprintf(report, "\n\n");
fprintf(report, "*****\n");
fprintf(report, "\tCONGRATULATIONS - %s has been successfully compiled.\n", GroupName);
}
else
{
    fprintf(report, "\n\n");
    fprintf(report, "*****\n");
    fprintf(report, "\tUnable to compile - Please notify developers\n");
}
}

```

```
}
else
{
    fprintf(report, "\n\n");
    fprintf(report, "*****\n");
    fprintf(report, "\tCompilation not attempted - No comps installed.\n");
}
fprintf(report, "*****\n");
fclose(report);
save_CompInfo();

/*****
Combine the date, install report, and compile report into one
master <group_name>.rpt
*****/
system("date '+DATE: %h %d, 19%y%TIME: %H:%M:%S%' > /tmp/date 2>> /tmp/code.err");
sprintf(message, "cat /tmp/date %s/%s/install.rpt %s/%s/compile.rpt >%s/%s/%s.rpt 2>>/tmp/code.err", CodeGrou
ps, GroupName, CodeGroups, GroupName, CodeGroups, GroupName, GroupName);
if(system(message) != 0)
{
    ask("Install: Unable to create install/compile report. Hit <RETURN>", RED, reply);
}

/*****
Write some blank lines to the end of the report,
so we can see error messages when we "more" the report.
*****/
sprintf(groupRpt, "%s/%s/%s.rpt", CodeGroups, GroupName, GroupName);
if (!(ptr1 = fopen (groupRpt, "a")))
{
    ask("Install: Unable to write group report.", RED, reply);
}
else
{
    for(i=0; i<60; i++)
        fprintf(ptr1, "\n");
    fclose(ptr1);
}
chmod(666, groupRpt);
/*****
Write the report to the work area window. Flush the input buffer that may have
been filled with garbage while input was not expected, and go into orig
tty mode.
*****/
ioctl (fileno (stdout), TCFLSH, 0);
ioctl (fileno (stdout), TCSETAW, &orig_tty);
/*****
Show the "MORE" prompt and select the work area
window for the output of the system call.
*****/
inform(MORE, GREEN, 0);
clearWA();
sprintf(message, "more -n30 %s/%s/%s.rpt ", CodeGroups, GroupName, GroupName);
if(system(message) != 0)
{
    reportError = TRUE;
    ask("Install: Unable to show install/compile report. Hit <RETURN>", RED, reply);
}
/*****
Go back to raw tty mode and put the comp back in the work area.
*****/
ioctl (fileno (stdout), TCSETAW, &raw_tty);
clearWA();
displayWA(Comp);

/*****
Print the report if desired.
*****/
sprintf(message, "Print report? (Y/N)");
ask(message, GREEN, reply);

if(reply[0] == 'Y' || reply[0] == 'y')
{
    sprintf(message, "print %s/%s/%s.rpt 2>>/tmp/code.err", CodeGroups, GroupName, GroupName);
    if(system(message) != 0)
    {
        ask("Install: Unable to create install/compile report. Hit <RETURN>", GREEN, message);
    }
}
```

```

read_vars(compName, this_comps_vars, comp_var_count)
char compName[];
struct var_struct this_comps_vars[];
int *comp_var_count;
{
    FILE *var_file;

    char variable_file[PATH_LEN];

    int index, /* The index returned from variable_exists if it exists */
        status, /* The status of the config_mgmt */
        i=0;
    sprintf (variable_file, "%s/%s.%v", CodeGroups, GroupName, compName);
    if (!(var_file = openFile (variable_file, "r","install")))
    {
        fprintf(report, "Cannot find variables for %s.\n", variable_file);
        return (ERROR);
    }

    /*****
     First we will discard the comment
     that is the 1st line of all variable files.
     *****/
    fscanf(var_file, "%*[^\n]");
    /*****/
    Now let's read the variable info for this
    comp until we reach the end of the file.
    *****/
    *comp_var_count = 0;
    while(fscanf (var_file, "%s %s %s %d %d %f %f %f %f %f [^\n]",
        this_comps_vars[*comp_var_count].name,
        this_comps_vars[*comp_var_count].type,
        this_comps_vars[*comp_var_count].class,
        &this_comps_vars[*comp_var_count].occurrence,
        &this_comps_vars[*comp_var_count].put_or_get,
        &this_comps_vars[*comp_var_count].lo1_limit,
        &this_comps_vars[*comp_var_count].lo2_limit,
        &this_comps_vars[*comp_var_count].hi1_limit,
        &this_comps_vars[*comp_var_count].hi2_limit,
        this_comps_vars[*comp_var_count].nomenclature) != EOF)
    {
        /*****/
        Let's do some CM if it's an MSID or a signal.
        *****/
        if(strcmp(this_comps_vars[*comp_var_count].class, "msid") == 0 ||
            strcmp(this_comps_vars[*comp_var_count].class, "signal") == 0)
        {
            config_mgmt(this_comps_vars[*comp_var_count].name,
                this_comps_vars[*comp_var_count].class,
                this_comps_vars[*comp_var_count].type,
                this_comps_vars[*comp_var_count].nomenclature);
        }

        /*****/
        First we need to see if the variable already
        exists in the group_var structure. If it doesn't
        exist we will increment the NumGroupVars and
        copy the *comp_var into the GroupVars list.
        If it existed we increment the occurrence
        field of the group_var
        *****/
        if ((index = variable_exists(this_comps_vars[*comp_var_count].name,
            this_comps_vars[*comp_var_count].type,
            this_comps_vars[*comp_var_count].class)) == ERROR)
        {
            GroupVars[NumGroupVars] = this_comps_vars[*comp_var_count];
            /*****/
            Let's keep track of how many msid's we have
            because we need to "#define MSID_CNT" in
            the <group>.c file.
            *****/
            if(strcmp(GroupVars[NumGroupVars].class, "msid") == 0)
                num_group_msids++;

            NumGroupVars++;
        }
    }
}

```


90/06/07
16:34:13

install.c

10

```
else GroupVars[index].occurrence++;  
    (*comp_var_count)++;  
}  
fclose (var_file);  
return(0);  
}
```

90/06/07
16:34:13

install.c

11

```
variable_exists(name, type, class)
char name[];
char type[];
char class[];
{
    int i;
    char temp[100];

    for(i=0; i<NumGroupVars;i++)
        if (strcmp(GroupVars[i].name, name) == 0)
        {
            return(i);
        }
    return(ERROR);
}
```

```
save_CompInfo()
{
    FILE      *CompInfo_file;

    char CompInfo_name[PATH_LEN];

    int i; /* Simple index */

    sprintf (CompInfo_name, "%s/%s.dat", AMSupport, GroupName);
    /*****
    Open group info file and write variables
    *****/
    if (!(CompInfo_file = openFile (CompInfo_name,"w","install")))
        return (ERROR);

    /*****
    Write all of the group info for the group with header.
    *****/
    fprintf (CompInfo_file, "#name_name noise_filter rate on_off disposition\n");
    for (i=0;i < NumOfComps;i++)
    {
        fprintf (CompInfo_file,"%s %d %d %d %d %s\n", CompInfo[i].name,
                    CompInfo[i].noise_filter,
                    CompInfo[i].rate,
                    CompInfo[i].on_off,
                    CompInfo[i].disposition,
                    CompInfo[i].purpose);
    }

    fclose(CompInfo_file);
}
```

```
write_comp(name)
char name[];
{
    FILE *ptr;

    char the_file[PATH_LEN], ch;

    sprintf(the_file, "%s/%s/%s.c", CodeGroups, GroupName, name);

    if (!(ptr = openFile(the_file, "r", "install")))
    {
        fprintf(report, "write_comp: unable to open %s.\n", the_file);
        return (ERROR);
    }

    while ( (ch = fgetc (ptr)) != EOF)
    {
        fputc (ch, group_file);
    }

    fputc ('\n', group_file);
    fputc ('\n', group_file);
    fclose(ptr);
}
```

```
config_mgmt(name, class, type, nomenclature)
char name[];
char class[];
char type[];
char nomenclature[];
{
    int i;
    char compare_type[TYPE_LEN]; /* the corrected type from the signal table 'S' -> 'c' */

    /*****
     * If it's a signal make sure is in the SignalTable list
     *****/
    if(strcmp(class, "signal") == 0)
    {
        for (i=0; i < SignalCount; i++)
        {
            if (strcmp (name, SignalTable[i].name) == 0)
            {
                if (SignalTable[i].type[0] == 'S')
                    strcpy(compare_type, "c");
                else strcpy (compare_type, SignalTable[i].type);
                /*****
                 * It's in the signal table - verify
                 * that the type is correct.
                 *****/
                if(type[0] != compare_type[0])
                {
                    fprintf(report, "\t%s type is not consistent with signal table.\n", name);
                    return(ERROR);
                }
                else return(OK);
            }
        }
        /*****
         * It's not defined in the signal table.
         * See if it's already in our array of
         * signals that need to be defined.
         *****/
        for (i=0; i<numSigsTBDef; i++)
        {
            if(strcmp(sigsTBDef[i].name, name) == 0)
            {
                /*****
                 * If it's already in our list of signals to be defined
                 * make sure it has the same type and nomenclature.
                 *****/
                if(strcmp(sigsTBDef[i].type, type)==0 &&
                    strcmp(sigsTBDef[i].nomenclature, nomenclature)==0)
                {
                    return(OK);
                }
                else
                {
                    fprintf(report, "\t%s type is not consistent with previous definition.\n", name);
                    return(ERROR);
                }
            }
        }
        /*****
         * It's not in the list to be defined so put it there.
         *****/
        strcpy(sigsTBDef[numSigsTBDef].name, name);
        strcpy(sigsTBDef[numSigsTBDef].type, type);
        strcpy(sigsTBDef[numSigsTBDef].nomenclature, nomenclature);
        numSigsTBDef++;
        return(OK);
    }
    else if (strcmp(class, "msid") == 0)
    {
        for (i=0; i < MSIDCount; i++)
        {
            if (strcmp (name, MSIDTable[i].name) == 0)
            {
                if(strcmp(type, MSIDTable[i].type) != 0)
                {
                    fprintf(report, "\t%s type is not consistent with tag_msid_table.\n", name);
                    return(ERROR);
                }
            }
        }
    }
}
```

90/06/07
16:34:13

install.c

15

```
    }  
    else return(OK);  
}  
}  
fprintf(report, "%t% is not defined in the tag_msid_table.\n", name);  
return(ERROR);  
}  
}
```

```

/*****
    compile_group

```

Purpose: Compile_group does a system call on cc, compiling the
the C file in order to create an executable. It
compiles with the correct libraries and include files.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 8/18/87

Version: 1.0

Project: CODE (Comp Development Environment)

```

/*****
compile_group ()
{
    char    sys_cmd[500],          /* String to hold system command. */
            compileRpt[PATH_LEN], /* String to hold file name of compile report */
            response[5],           /* A place for get user input */
            libPath[PATH_LEN];     /* string to build path to the libraries */

    int     i,                    /* A simple counter */
            rc;                   /* Return code for system command. */

    /*
     * Compile the group and redirect errors to the installation report
     * <GroupName>.rpt.
     */
    inform ("Please wait, compiling...", PURPLE, 0);

    sprintf(compileRpt, "%s/%s/compile.rpt", CodeGroups, GroupName);
    sprintf(libPath, "%s/rtds/lib", RTDS);
    sprintf (sys_cmd, "cc -o %s/%s %s/%s/%s.c -I$RTDS/rtds/include %s/libuserfuncs.a %s/libnf.a %s/libfltmsg.a %s/librti.a %s/libiesputil.a 2> %s", AMGroups, GroupName, CodeGroups, GroupName, GroupName, libPath, libPath, libPath, libPath, compileRpt);
    rc = system (sys_cmd);

    /*
     * If there were compilation errors return an ERROR.
     */
    if(rc != 0)
    {
        GroupInfo[GroupNumber].disposition = ERROR;
        /*
         * Put the group names into the GroupInfo file
         */
        writeGroupNames();
        return(ERROR);
    }
    else
    {
        GroupInfo[GroupNumber].disposition = INSTALLED;
        /*
         * Put the group names into the GroupNames file
         */
        writeGroupNames ();
        return(OK);
    }
}

```

90/06/07
16:36:47

menu.c

1

```

/*****
menu.c
*****/

#include "code.h"
#include "color.h"
int button_pressed; /* The number of the pressed button 1,2, or 4 */
/*****
The option_struct is used to store the mouse sensitive options and
there corresponding coordinates.
*****/
struct optionStruct
{
    char *name;
    float lowX;
    float lowY;
    float highX;
    float highY;
};

struct optionStruct options[NUMBER_OF_OPTIONS] = {
    "if",      0.0081, 0.8540, 0.0681, 0.8790,
    "then",    0.0844, 0.8540, 0.1444, 0.8790,
    "and",     0.0081, 0.7950, 0.0681, 0.8200,
    "or",      0.0844, 0.7950, 0.1444, 0.8200,
    "not",     0.0844, 0.7655, 0.1444, 0.7905,
    "print",   0.0081, 0.2820, 0.0681, 0.3070,
    "set",     0.0081, 0.3115, 0.0681, 0.3365,
    ">",       0.1101, 0.3735, 0.1201, 0.3985,
    ">=",     0.0590, 0.3735, 0.0790, 0.3985,
    "<",       0.1101, 0.4030, 0.1201, 0.4280,
    "<=",     0.0590, 0.4030, 0.0790, 0.4280,
    "=",       0.0217, 0.4030, 0.0317, 0.4280,
    "MSID",    0.0081, 0.6450, 0.0681, 0.6700,
    "signal",  0.0844, 0.6450, 0.1444, 0.6700,
    "local",   0.0081, 0.6155, 0.0681, 0.6405,
    "number",  0.0844, 0.6155, 0.1444, 0.6405,
    "integer", 0.0794, 0.5240, 0.1494, 0.5490,
    "float",   0.0081, 0.4945, 0.0681, 0.5195,
    "double",  0.0844, 0.4945, 0.1444, 0.5195,
    "New",     0.0081, 0.9455, 0.0681, 0.9705,
    "Save",    0.2500, 0.9455, 0.2900, 0.9705,
    "Install", 0.3000, 0.9455, 0.3700, 0.9705,
    "Retrieve", 0.1600, 0.9455, 0.2400, 0.9705,
    "Quit",    0.6600, 0.9455, 0.7000, 0.9705,
    "Delete",  0.0081, 0.9160, 0.0681, 0.9410,
    "else",    0.0081, 0.8245, 0.0681, 0.8495,
    "endif",   0.0844, 0.8245, 0.1444, 0.8495,
    "List",    0.7637, 0.9455, 0.8050, 0.9705,
    "8",       0.0217, 0.3735, 0.0317, 0.3985,
    "(",       0.0155, 0.1905, 0.0255, 0.2155,
    ")",       0.0525, 0.1905, 0.0625, 0.2155,
    "+",       0.0155, 0.2200, 0.0255, 0.2450,
    "-",       0.0525, 0.2200, 0.0625, 0.2450,
    "**",      0.0895, 0.2200, 0.0995, 0.2450,
    "/",      0.1265, 0.2200, 0.1365, 0.2450,
    "Remove",  0.4300, 0.9455, 0.4900, 0.9705,
    "Edit",    0.0844, 0.9455, 0.1444, 0.9705,
    "comment", 0.0720, 0.2820, 0.1520, 0.3070,
    "Hardcopy", 0.5000, 0.9455, 0.5800, 0.9705,
    "bitXor",  0.0081, 0.7655, 0.0681, 0.7905,
    "Backup",  0.5900, 0.9455, 0.6500, 0.9705,
    "short",   0.0081, 0.5240, 0.0681, 0.5490,
    "char",    0.0081, 0.4650, 0.0681, 0.4900, /* Type's string */
    "function", 0.0720, 0.3115, 0.1520, 0.3365,
    "exp",     0.0155, 0.1315, 0.0455, 0.1565,
    "log",     0.0895, 0.1315, 0.1095, 0.1565,
    "cos",     0.0155, 0.0695, 0.0681, 0.0945,
    "acos",    0.0895, 0.0695, 0.1444, 0.0945,
    "sin",     0.0155, 0.0400, 0.0681, 0.0650,
    "asin",    0.0895, 0.0400, 0.1444, 0.0650,
    "tan",     0.0155, 0.0105, 0.0681, 0.0355,
    "atan",    0.0895, 0.0105, 0.1444, 0.0355,
    "Copy",    0.3800, 0.9455, 0.4200, 0.9705,
    "sqrt",    0.0155, 0.1610, 0.0555, 0.1860,
    "power",   0.0895, 0.1610, 0.1495, 0.1860,
    "string",  0.0081, 0.5860, 0.0681, 0.6110, /* Variable's string */
    "bitAnd",  0.0081, 0.7360, 0.0681, 0.7610,
    "bitOr",   0.0844, 0.7360, 0.1444, 0.7610,

```



```
"shiftL", 0.0155, 0.1020, 0.0755, 0.1270,
"shiftR", 0.0895, 0.1020, 0.1495, 0.1270,
"PI",      0.0895, 0.1905, 0.0995, 0.2155,
",",       0.1265, 0.1905, 0.1365, 0.2155,
"Help",    0.7187, 0.9455, 0.7587, 0.9705,
"Small",   0.8200, 0.9455, 0.8600, 0.9705,
"Medium",  0.8750, 0.9455, 0.9150, 0.9705,
"Large",   0.9400, 0.9455, 0.9800, 0.9705,
};

struct helpOptionStruct
{
    int name;
    float lowX;
    float lowY;
    float highX;
    float highY;
};

struct helpOptionStruct help_options[NUMBER_OF_HELP] = {
    WORK_AREA, 0.1563, 0.1833, 1.0000, 0.9110,
    KEYIN_AREA, 0.1563, 0.0850, 1.0000, 0.1600,
    PROMPT_AREA, 0.1563, 0.0000, 1.0000, 0.0600
};
```

```

menu (theMode)
int theMode;
{
/*****
    Local Variables
*****/
options      all the possible options to mouse on
x_picked     x coordinate of mouse location
y_picked     y coordinate of mouse location
i            simple counter
j            simple counter
*****/

int    pushed(),      /* The routine down below called by mgibuttonint */
    show_buf,        /* The graphics buffer to show */
    mod_buf,         /* The graphics buffer to modify */
    i;               /* simple counter */

char temp[150];      /* always need one of these */

float x_picked, y_picked; /* relative coordinates of mouse */

/*****
    Assume a button has not been pressed
*****/
button_pressed = 0;
mgibuttonint (0);

/*****
    Hang out until a button is pressed
*****/
mgibuttonint (pushed);
while (!(button_pressed))
    astpause (-1, 100);

inform("", BLUE, 0);

/*****
    Get the relative coordinates of mouse
*****/
mgrgetcursxy (2, 1, &x_picked, &y_picked);

mgibuttonint (0);

/*****
    Find out what area we are in
*****/
if (theMode == HELP || theMode == LIST)
{
    for (i=0; i<NUMBER_OF_OPTIONS; i++)
    {
/*****
        Have they chosen a token
*****/
        if (x_picked < options[i].highX &&
            x_picked > options[i].lowX &&
            y_picked < options[i].highY &&
            y_picked > options[i].lowY)
        {
            mgifb(1, 1);
            return (i);
        }
    }
    inform("You are not over a mouseable token.", BLUE, 0);
    /*
     * Show the frame buffer we're not currently showing.
     * Modify the frame buffer we're not currently modifying.
     */
    mgigetfb (&show_buf, &mod_buf);
    if (show_buf==1)
        clearWA();
    else displayWA(Comp);
    mgifb (11, 1);
    return(ERROR);
}
else if (theMode == FONT)
{

```

```

/*****
See if we're mousing in one of our windows
*****/
for (i=0;i<NUMBER_OF_HELP;i++)
{
    /*****
    Have they chosen a token
    *****/
    if (x_picked < help_options[i].highX &&
        x_picked > help_options[i].lowX &&
        y_picked < help_options[i].highY &&
        y_picked > help_options[i].lowY)
    {
        mgifb(1,1);
        return (help_options[i].name);
    }
}
inform("You are not over a mouseable token.",BLUE,0);
/*
 * Show the frame buffer we're not currently showing.
 * Modify the frame buffer we're not currently modifying.
 */
mgigetfb (&show_buf, &mod_buf);
if (show_buf==1)
    clearWA();
else displayWA(Comp);
mgifb (11, 1);
return(ERROR);
}
else if (theMode == RUN)
{
    for (i=0;i<TotValPts;i++)
    {
        if (x_picked < options[ValidPoints[i]].highX &&
            x_picked > options[ValidPoints[i]].lowX &&
            y_picked < options[ValidPoints[i]].highY &&
            y_picked > options[ValidPoints[i]].lowY)
        {
            mgifb(1,1);
            return(ValidPoints[i]);
        }
    }
}
inform("You are not over a mouseable token.",BLUE,0);
/*
 * Show the frame buffer we're not currently showing.
 * Modify the frame buffer we're not currently modifying.
 */
mgigetfb (&show_buf, &mod_buf);
if (show_buf==1)
    clearWA();
else displayWA(Comp);
mgifb (11, 1);
return(ERROR);
}
}

```

/*****

COLOR_VALID

Purpose: Color valid is used to write all the valid next choices
in a different color (yellow) than the invalid choices
(red).

Designer: J. Harold Taylor/SDC

Programmer: J. Harold Taylor/SDC

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

None.

*****/

color_valid (previous, other)

/*****

Global Variables

*****/

int previous, other;

{

/*****

Local Variables

*****/

options all the possible options to mouse on
low_x_point the lower left x coordinate of the mousable area
low_y_point the lower left y coordinate of the mousable area
high_x_point the upper right x coordinate of the mousable area
high_y_point the upper right y coordinate of the mousable area
i simple counter
j simple counter
flag flag for determining color of this particular choice

*****/

int i;
char temp[100];

/*****

Call next_inputs if required

*****/

next_inputs (previous, other);

if (previous < 0 || previous > 50)
 previous = 0;

mgihue(RED);
for (i=0;i<NUMBER_OF_OPTIONS;i++)

{
 /*****
 Draw the allowable options given that we need to
 size these normalized coordinates based on screen resolution
 *****/
 mgrgfs (options[i].lowX + 0.0024,options[i].lowY + 0.0012, 0, options[i].name);
}

/*****
Now go through and draw all the valid tokens in yellow
*****/
mgihue(YELLOW);
for (i=0;i<TotValPts;i++)
{

90/06/07
16:36:47

menu.c

6

```

/*****
Draw the allowable options given that we need to
size these normalized coordinates based on screen resolution
*****/
mrgqfs (options[ValidPoints[i]].lowX + 0.0024,
        options[ValidPoints[i]].lowY + 0.0012, 0,
        options[ValidPoints[i]].name);
}

mgicursmode (9);
mgrcursxy (2,6,options[LikelyNextChoice].lowX + 0.0048,
          options[LikelyNextChoice].lowY + 0.013);
mgicursmode (15);
}

```

```
/******  
    PUSHED
```

Purpose: Pushed is simply the routine that is invoked when
a mouse button is pressed. It must be preceded by
a mgibuttonint (pushed) command

Designer: J. Harold Taylor/SDC

Programmer: J. Harold Taylor/SDC

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

None.

```
*****  
    Global Variables  
*****  
    x      x coordinate of cursor  
    y      y coordinate of cursor  
    s      mask for which button has been pressed  
*****/  
pushed (x, y, s)
```

unsigned int x,y,s;

```
{  
    if( s )  
    {  
        button_pressed = s;  
    }  
}
```

90/06/07
16:37:11

new.c

1

/******

NEW

Purpose: Prompts the user for the name of a new group or comp.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 2/8/89

Version: 2.0

Project: CODE (Comp Development Environment)

*****/

#include "code.h"

#include "color.h"

new(theClass)

char theClass[]; /* Either "Comp" or "Group" */

{

char tempStr[200], nameToGet[MAX_NAME_LEN], group_dir[PATH_LEN];

int i, status;

/*

* Clean the mean screen in prep for the new comp.

*/

cleanSlate();

mgihue (BLACK); /* Blank out the out names */

mgrbox (.7, 0.9120, .999, 0.9320);

/*

* Get the new name from the user, if the user defaults

* we return to calling routine with no changes.

*/

if (get_new_name(theClass, nameToGet) == DEFAULT)

return(DEFAULT);

if (!strcmp(theClass, "Comp"))

{

CompNumber = NumOfComps++;

strcpy (CompName, nameToGet);

strcpy (CompInfo[CompNumber].name, CompName);

CompInfo[CompNumber].disposition = INCOMPLETE;

CompInfo[CompNumber].rate = 1;

CompInfo[CompNumber].on_off = FALSE;

CompInfo[CompNumber].noise_filter = 2;

putStrWithBlue(CompName, 0.8823); /* string, norm x */

}

else /* It was a group */

{

GroupNumber = NumOfGroups;

strcpy (GroupName, nameToGet);

strcpy (GroupInfo[NumOfGroups].name, GroupName);

/*

* This is the only place where NumOfGroups++

*/

GroupInfo[NumOfGroups++].disposition = INCOMPLETE;

Create the necessary directory since it does not exist yet.

*****/

sprintf (group_dir, "%s/%s", CodeGroups, GroupName);

mkdir (group_dir, 511);

NumOfComps = 0;

putStrWithBlue(GroupName, 0.75); /* string, norm x */

}

}

get_new_name(theClass, nameToGet)

char theClass[], /* Whether it is a 'Comp' or 'Group' */

nameToGet; / The name we shall set within */

char tmpStr[200];

int i, status;

do

```
(
    sprintf(tmpStr, "Please type the new %s name(<%d char), or (Q)uit", theClass, MAX_NAME_LEN-2);
    status = ask(tmpStr, GREEN, nameToGet);
    if (status == DEFAULT || nameToGet[0] == 'q' || nameToGet[0] == 'Q')
        return(DEFAULT);
    /*****
    File names may not have a leading zero
    *****/
    if (nameToGet[0] >= '0' && nameToGet[0] <= '9')
    {
        status = ERROR;
        ask("First character must be alphabetic -- Hit [RETURN] to continue", GREEN, tmpStr);
    }
    /*****
    Make sure that it's not an already existing name.
    *****/
    else
    {
        /*
        * The comp name is called as a subroutine
        * and therefore can not have the name main.
        */
        if (!strcmp("main", nameToGet))
        {
            status = ERROR;
            inform("'main' is reserved; do not use it.", PURPLE, 2);
        }
        else if (!strcmp(theClass, "Group"))
        {
            /*
            * Look for a duplicate group name
            */
            for (i=0; i<NumOfGroups; ++i)
            {
                if (!strcmp (GroupInfo[i].name, nameToGet))
                {
                    status = ERROR;
                    sprintf(tmpStr, "%s already exists. - Hit [RETURN] to continue", nameToGet);
                    ask(tmpStr, GREEN, tmpStr);
                    break;
                }
            }
        }
        else /* Look through the list of comps */
        {
            /*
            * Look for a duplicate comp name
            */
            for (i=0; i<NumOfComps; ++i)
            {
                if (!strcmp (CompInfo[i].name, nameToGet))
                {
                    status = ERROR;
                    sprintf(tmpStr, "%s already exists -- Hit [RETURN] to continue", nameToGet);
                    ask (tmpStr, GREEN, tmpStr);
                    break;
                }
            }
        }
    }
} while (status != OK);
return OK;
}
```


90/06/07
16:39:28

next_inputs.c

1

NEXT_INPUTS

Purpose: Next_Inputs is used to determine what options are valid after each choice.

Designer: J. Harold Taylor/SDC

Programmer: J. Harold Taylor/SDC

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

External Interfaces

```

/*****
*****/

/*****
***** Include files
*****/

#include "code.h"

next_inputs (on_my_left,on_my_right)

int on_my_left,on_my_right;

{
    int i, defaults = 17;
    char temp[156];

    if (on_my_right == -2)
        return (OK);

    TotValPts = defaults;

    /* The following 17 points are default and should always be mouseable */
    ValidPoints[0] = QUIT;
    ValidPoints[1] = HELP;
    ValidPoints[2] = RETRIEVE;
    ValidPoints[3] = HARDCOPY;
    ValidPoints[4] = EDIT;
    ValidPoints[5] = SAVE;
    ValidPoints[6] = REMOVE;
    ValidPoints[7] = INSTALL;
    ValidPoints[8] = COMMENT;
    ValidPoints[9] = DELETE;
    ValidPoints[10] = NEW;
    ValidPoints[11] = COPY;
    ValidPoints[12] = LIST;
    ValidPoints[13] = SMALL;
    ValidPoints[14] = MEDIUM;
    ValidPoints[15] = LARGE;
    ValidPoints[16] = BACKUP;

    /*****
    ***** COMMENTS
    *****
    Comments are handled differently than other tokens, since they do
    not change what is valid, but are part of the comp string. Search
    backward for a valid token, or just find the end of the string.
    *****/
    if (on_my_left == COMMENT)
    {
        /*****
        ***** Start out assuming its a new comp
        *****/
        on_my_left = NEW;
        /*****
        ***** Loop through the choices until we find one which
        is not a comment, then reset and break
        *****/
    }
}

```

```

*****
for (i=ChoiceCounter-1;i>1;i--)
(
    /*****
    Once we have found the first occurrence of
    something that is not a comment we call, break
    *****/
    if (PrevChoice[i] != COMMENT)
    {
        on_my_left = PrevChoice[i];
        break;
    }
)
)
/*****
    IF
*****/
if (on_my_left == IF)
(
    TotValPts = defaults + 20;
    ValidPoints[defaults + 0] = LOCAL;
    ValidPoints[defaults + 1] = MSID;
    ValidPoints[defaults + 2] = SIGNAL;
    ValidPoints[defaults + 3] = NUMBER;
    ValidPoints[defaults + 4] = L_PAREN;
    ValidPoints[defaults + 5] = NOT;
    ValidPoints[defaults + 6] = COS;
    ValidPoints[defaults + 7] = ACOS;
    ValidPoints[defaults + 8] = SIN;
    ValidPoints[defaults + 9] = ASIN;
    ValidPoints[defaults + 10] = TAN;
    ValidPoints[defaults + 11] = ATAN;
    ValidPoints[defaults + 12] = STRING;
    ValidPoints[defaults + 13] = PI;
    ValidPoints[defaults + 14] = FUNCTION;
    ValidPoints[defaults + 15] = SQRT;
    ValidPoints[defaults + 16] = POWER;
    ValidPoints[defaults + 17] = EXP;
    ValidPoints[defaults + 18] = LOG;
    ValidPoints[defaults + 19] = SUBTRACT;
    LikelyNextChoice = MSID;
)
/*****
    GET_TYPE
    If the user is to enter the type, don't let
    do anything else until they do.
*****/
else if (on_my_left == GET_TYPE)
(
    TotValPts = 5;
    ValidPoints[0] = FLOAT;
    ValidPoints[1] = DOUBLE;
    ValidPoints[2] = INTEGER;
    ValidPoints[3] = SHORT;
    ValidPoints[4] = CHAR;
    LikelyNextChoice = SHORT;
)
/*****
    MSID NUMBER PI CHAR SIGNAL LOCAL
*****/
else if (on_my_left == MSID || on_my_left == NUMBER ||
    on_my_left == PI || on_my_left == STRING ||
    on_my_left == SIGNAL || on_my_left == LOCAL)
(
    /*****
    Having an equal number of ifs and endifs means
    we're outside of the if which is the same
    as being in the consequence.
    *****/
    if (WhereAmI == CONSEQUENCE || NumberOfIfs == NumberOfEndifs)
    (
        if (Equation == RHS)
        {
            if (CompareType[NumberOfCompares-1][0] != 'c')
            {
                /*****
                Check if they have balanced their paren's,
                if not, make 'em balance the parens.
                *****/
            }
        }
    )
)

```

```

This takes care of argDefs too.
*****/
if (ParenCount > 0)
{
    TotValPts = defaults + 10;
    ValidPoints[defaults + 0] = ADD;
    ValidPoints[defaults + 1] = SUBTRACT;
    ValidPoints[defaults + 2] = MULTIPLY;
    ValidPoints[defaults + 3] = DIVIDE;
    ValidPoints[defaults + 4] = SHIFTR;
    ValidPoints[defaults + 5] = SHIFTL;
    ValidPoints[defaults + 6] = R_PAREN;
    ValidPoints[defaults + 7] = BITOR;
    ValidPoints[defaults + 8] = BITXOR;
    ValidPoints[defaults + 9] = BITAND;
    LikelyNextChoice = R_PAREN;
    if(FuncParenCount > 0 &&
        FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
        ValidPoints[TotValPts++] = COMMA;
}
else
{
    TotValPts = defaults + 12;
    ValidPoints[defaults + 0] = ADD;
    ValidPoints[defaults + 1] = SUBTRACT;
    ValidPoints[defaults + 2] = MULTIPLY;
    ValidPoints[defaults + 3] = DIVIDE;
    ValidPoints[defaults + 4] = IF;
    ValidPoints[defaults + 5] = SET;
    ValidPoints[defaults + 6] = PRINT;
    ValidPoints[defaults + 7] = SHIFTR;
    ValidPoints[defaults + 8] = SHIFTL;
    ValidPoints[defaults + 9] = BITOR;
    ValidPoints[defaults + 10] = BITXOR;
    ValidPoints[defaults + 11] = BITAND;
    LikelyNextChoice = IF;
    /*****
     * If an ELSE has already been used in this if
     * we don't let them use it again!
     *****/
    if(NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    {
        LikelyNextChoice = ELSE;
        ValidPoints[TotValPts++] = ELSE;
    }
    /*****
     * If they have closed all their if's with
     * endif's then they don't need endif any more.
     *****/
    if ((NumberOfIfs - NumberOfEndifs) > 0)
    {
        LikelyNextChoice = END_IF;
        ValidPoints[TotValPts++] = END_IF;
    }
}
/*
 * Compare type is a string.
 */
else
{
    TotValPts = defaults + 3;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SET;
    ValidPoints[defaults + 2] = PRINT;
    /*****
     * If an ELSE has already been used in this if
     * we don't let them use it again!
     *****/
    if(NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    {
        LikelyNextChoice = ELSE;
        ValidPoints[TotValPts++] = ELSE;
    }
    /*****
     * If they have closed all their if's with
     * endif's then they don't need endif any more.
     *****/
}

```

```
    if ((NumberOfIfs - NumberOfEndifs) > 0)
    {
        LikelyNextChoice = END_IF;
        ValidPoints[TotValPts++] = END_IF;
    }
}
/*****
Don't worry - be happy --- we couldn't get to
this point with an MSID or constant expression.
See on_my_left == SET.
*****/
else if (Equation == LHS)
{
    TotValPts = defaults + 1;
    ValidPoints[defaults + 0] = EQ;
    LikelyNextChoice = EQ;
}
}
else if (WhereAmI == PREMISE)
{
    if (Equation == LHS)
    {
        if (ParenCount > 0)
        {
            /*****
            If we're done defining arguments, let
            them do some relational stuff.
            *****/
            if (FuncParenCount == 0)
            {
                TotValPts = defaults + 16;
                ValidPoints[defaults + 0] = ADD;
                ValidPoints[defaults + 1] = SUBTRACT;
                ValidPoints[defaults + 2] = MULTIPLY;
                ValidPoints[defaults + 3] = DIVIDE;
                ValidPoints[defaults + 4] = LT;
                ValidPoints[defaults + 5] = GT;
                ValidPoints[defaults + 6] = LE;
                ValidPoints[defaults + 7] = GE;
                ValidPoints[defaults + 8] = NE;
                ValidPoints[defaults + 9] = R_PAREN;
                ValidPoints[defaults + 10] = EQ;
                ValidPoints[defaults + 11] = SHIFTR;
                ValidPoints[defaults + 12] = SHIFTL;
                ValidPoints[defaults + 13] = BITOR;
                ValidPoints[defaults + 14] = BITXOR;
                ValidPoints[defaults + 15] = BITAND;

                LikelyNextChoice = EQ;
            }
            else
            {
                TotValPts = defaults + 10;
                ValidPoints[defaults + 0] = ADD;
                ValidPoints[defaults + 1] = SUBTRACT;
                ValidPoints[defaults + 2] = MULTIPLY;
                ValidPoints[defaults + 3] = DIVIDE;
                ValidPoints[defaults + 4] = SHIFTR;
                ValidPoints[defaults + 5] = SHIFTL;
                ValidPoints[defaults + 6] = R_PAREN;
                ValidPoints[defaults + 7] = BITOR;
                ValidPoints[defaults + 8] = BITXOR;
                ValidPoints[defaults + 9] = BITAND;
                if (FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
                    ValidPoints[TotValPts++] = COMMA;
                LikelyNextChoice = R_PAREN;
            }
        }
        else
        {
            TotValPts = defaults + 15;
            ValidPoints[defaults + 0] = ADD;
            ValidPoints[defaults + 1] = SUBTRACT;
            ValidPoints[defaults + 2] = MULTIPLY;
            ValidPoints[defaults + 3] = DIVIDE;
            ValidPoints[defaults + 4] = LT;
            ValidPoints[defaults + 5] = GT;
```

next_inputs.c

```

ValidPoints[defaults + 6] = LE;
ValidPoints[defaults + 7] = GE;
ValidPoints[defaults + 8] = NE;
ValidPoints[defaults + 9] = EQ;
ValidPoints[defaults + 10] = SHIFTR;
ValidPoints[defaults + 11] = SHIFTL;
ValidPoints[defaults + 12] = BITOR;
ValidPoints[defaults + 13] = BITXOR;
ValidPoints[defaults + 14] = BITAND;
LikelyNextChoice = EQ;
    }
}
else if (Equation == RHS)
{
    if (ParenCount > 0)
    {
        /*****
        If we're done defining function arguments,
        let them do some relational stuff.
        *****/
        if (FuncParenCount == 0)
        {
            TotValPts = defaults + 15;
            ValidPoints[defaults + 0] = AND;
            ValidPoints[defaults + 1] = DIVIDE;
            ValidPoints[defaults + 2] = OR;
            ValidPoints[defaults + 3] = ADD;
            ValidPoints[defaults + 4] = SUBTRACT;
            ValidPoints[defaults + 5] = MULTIPLY;
            ValidPoints[defaults + 6] = BITXOR;
            ValidPoints[defaults + 7] = R_PAREN;
            ValidPoints[defaults + 8] = BITAND;
            ValidPoints[defaults + 9] = BITOR;
            ValidPoints[defaults + 10] = SHIFTR;
            ValidPoints[defaults + 11] = SHIFTL;
            LikelyNextChoice = R_PAREN;
        }
        else
        {
            TotValPts = defaults + 10;
            ValidPoints[defaults + 0] = DIVIDE;
            ValidPoints[defaults + 1] = ADD;
            ValidPoints[defaults + 2] = SUBTRACT;
            ValidPoints[defaults + 3] = MULTIPLY;
            ValidPoints[defaults + 4] = R_PAREN;
            ValidPoints[defaults + 5] = SHIFTR;
            ValidPoints[defaults + 6] = SHIFTL;
            ValidPoints[defaults + 7] = BITOR;
            ValidPoints[defaults + 8] = BITXOR;
            ValidPoints[defaults + 9] = BITAND;

            if (FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
                ValidPoints[TotValPts++] = COMMA;
            LikelyNextChoice = R_PAREN;
        }
    }
    else
    {
        TotValPts = defaults + 12;
        ValidPoints[defaults + 0] = AND;
        ValidPoints[defaults + 1] = DIVIDE;
        ValidPoints[defaults + 2] = OR;
        ValidPoints[defaults + 3] = ADD;
        ValidPoints[defaults + 4] = SUBTRACT;
        ValidPoints[defaults + 5] = MULTIPLY;
        ValidPoints[defaults + 6] = BITXOR;
        ValidPoints[defaults + 7] = THEN;
        ValidPoints[defaults + 8] = BITAND;
        ValidPoints[defaults + 9] = BITOR;
        ValidPoints[defaults + 10] = SHIFTR;
        ValidPoints[defaults + 11] = SHIFTL;
        LikelyNextChoice = THEN;
    }
}
}
}
}
/*****
Logical ops covers <, >, <=, >=, =, <>
*****/

```

90/06/07
16:39:28

next_inputs.c

6

```
*****  
else if ((on_my_left == GT) || (on_my_left == LT) ||  
         (on_my_left == GE) || (on_my_left == LE) ||  
         (on_my_left == NE) || (on_my_left == EQ) ||  
         (on_my_left == ADD) || (on_my_left == SUBTRACT) ||  
         (on_my_left == MULTIPLY) || (on_my_left == DIVIDE) ||  
         (on_my_left == SHIFTL) || (on_my_left == SHIFTR) ||  
         (on_my_left == BITOR) || (on_my_left == BITAND))  
{  
    TotValPts = defaults + 8;  
    ValidPoints[defaults + 0] = MSID;  
    ValidPoints[defaults + 1] = SIGNAL;  
    ValidPoints[defaults + 2] = LOCAL;  
    ValidPoints[defaults + 3] = FUNCTION;  
    ValidPoints[defaults + 4] = NUMBER;  
    ValidPoints[defaults + 5] = L_PAREN;  
    ValidPoints[defaults + 6] = SUBTRACT;  
    ValidPoints[defaults + 7] = STRING;  
  
    if ((WhereAmI == CONSEQUENCE && CompareType[NumberOfCompares-1][0] == 'd') ||  
        WhereAmI == PREMISE)  
    {  
        ValidPoints[TotValPts++] = COS;  
        ValidPoints[TotValPts++] = ACOS;  
        ValidPoints[TotValPts++] = SIN;  
        ValidPoints[TotValPts++] = ASIN;  
        ValidPoints[TotValPts++] = TAN;  
        ValidPoints[TotValPts++] = ATAN;  
        ValidPoints[TotValPts++] = PI;  
        ValidPoints[TotValPts++] = SQRT;  
        ValidPoints[TotValPts++] = POWER;  
        ValidPoints[TotValPts++] = EXP;  
        ValidPoints[TotValPts++] = LOG;  
    }  
    LikelyNextChoice = NUMBER;  
}  
/*****  
    AND, OR, BITXOR, BITAND, BITOR  
*****/  
*****  
else if (on_my_left == OR || on_my_left == BITXOR || on_my_left == AND )  
{  
    TotValPts = defaults + 20;  
    ValidPoints[defaults + 0] = SIGNAL;  
    ValidPoints[defaults + 1] = LOCAL;  
    ValidPoints[defaults + 2] = MSID;  
    ValidPoints[defaults + 3] = L_PAREN;  
    ValidPoints[defaults + 4] = NOT;  
    ValidPoints[defaults + 5] = NUMBER;  
    ValidPoints[defaults + 6] = STRING;  
    ValidPoints[defaults + 7] = PI;  
    ValidPoints[defaults + 8] = COS;  
    ValidPoints[defaults + 9] = ACOS;  
    ValidPoints[defaults + 10] = SIN;  
    ValidPoints[defaults + 11] = ASIN;  
    ValidPoints[defaults + 12] = TAN;  
    ValidPoints[defaults + 13] = ATAN;  
    ValidPoints[defaults + 14] = FUNCTION;  
    ValidPoints[defaults + 15] = SQRT;  
    ValidPoints[defaults + 16] = POWER;  
    ValidPoints[defaults + 17] = EXP;  
    ValidPoints[defaults + 18] = LOG;  
    ValidPoints[defaults + 19] = SUBTRACT;  
    LikelyNextChoice = MSID;  
}  
/*****  
    THEN  
*****/  
*****  
else if (on_my_left == THEN)  
{  
    TotValPts = defaults + 3;  
    ValidPoints[defaults + 0] = IF;  
    ValidPoints[defaults + 1] = PRINT;  
    ValidPoints[defaults + 2] = SET;  
    LikelyNextChoice = PRINT;  
}  
/*****  
    ELSE  
*****/  
*****
```

```
else if (on_my_left == ELSE)
{
    TotValPts = defaults + 6;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SIGNAL;
    ValidPoints[defaults + 2] = LOCAL;
    ValidPoints[defaults + 3] = MSID;
    ValidPoints[defaults + 4] = PRINT;
    ValidPoints[defaults + 5] = SET;
    LikelyNextChoice = PRINT;
}
/*****
    NOT
*****/
else if (on_my_left == NOT)
{
    TotValPts = defaults + 4;
    ValidPoints[defaults + 0] = MSID;
    ValidPoints[defaults + 1] = SIGNAL;
    ValidPoints[defaults + 2] = LOCAL;
    ValidPoints[defaults + 3] = L_PAREN;
    LikelyNextChoice = MSID;
}
/*****
    L_PAREN
*****/
else if (on_my_left == L_PAREN)
{
    TotValPts = defaults + 20;
    ValidPoints[defaults + 0] = MSID;
    ValidPoints[defaults + 1] = SIGNAL;
    ValidPoints[defaults + 2] = LOCAL;
    ValidPoints[defaults + 3] = NUMBER;
    ValidPoints[defaults + 4] = STRING;
    ValidPoints[defaults + 5] = NOT;
    ValidPoints[defaults + 6] = PI;
    ValidPoints[defaults + 7] = COS;
    ValidPoints[defaults + 8] = ACOS;
    ValidPoints[defaults + 9] = SIN;
    ValidPoints[defaults + 10] = ASIN;
    ValidPoints[defaults + 11] = TAN;
    ValidPoints[defaults + 12] = ATAN;
    ValidPoints[defaults + 13] = FUNCTION;
    ValidPoints[defaults + 14] = SQRT;
    ValidPoints[defaults + 15] = POWER;
    ValidPoints[defaults + 16] = EXP;
    ValidPoints[defaults + 17] = LOG;
    ValidPoints[defaults + 18] = L_PAREN;
    ValidPoints[defaults + 19] = SUBTRACT;
    LikelyNextChoice = MSID;
}
/*****
    COMMA
*****/
else if (on_my_left == COMMA)
{
    TotValPts = defaults + 20;
    ValidPoints[defaults + 0] = MSID;
    ValidPoints[defaults + 1] = SIGNAL;
    ValidPoints[defaults + 2] = LOCAL;
    ValidPoints[defaults + 3] = NUMBER;
    ValidPoints[defaults + 4] = STRING;
    ValidPoints[defaults + 5] = NOT;
    ValidPoints[defaults + 6] = PI;
    ValidPoints[defaults + 7] = COS;
    ValidPoints[defaults + 8] = ACOS;
    ValidPoints[defaults + 9] = SIN;
    ValidPoints[defaults + 10] = ASIN;
    ValidPoints[defaults + 11] = TAN;
    ValidPoints[defaults + 12] = ATAN;
    ValidPoints[defaults + 13] = FUNCTION;
    ValidPoints[defaults + 14] = SQRT;
    ValidPoints[defaults + 15] = POWER;
    ValidPoints[defaults + 16] = EXP;
    ValidPoints[defaults + 17] = LOG;
    ValidPoints[defaults + 18] = L_PAREN;
    ValidPoints[defaults + 19] = SUBTRACT;
    LikelyNextChoice = MSID;
}
```

```

)
/*****
R_PAREN
*****/
else if (on_my_left == R_PAREN)
{
/*****
/ + - * SHIFTL SHIFTR BITOR BITXOR BITAND
are always allowed after a R_PAREN.
*****/
TotValPts = defaults + 10;
ValidPoints[defaults + 0] = DIVIDE;
ValidPoints[defaults + 1] = ADD;
ValidPoints[defaults + 2] = SUBTRACT;
ValidPoints[defaults + 3] = MULTIPLY;
ValidPoints[defaults + 4] = SHIFTR;
ValidPoints[defaults + 5] = SHIFTL;
ValidPoints[defaults + 7] = BITOR;
ValidPoints[defaults + 8] = BITXOR;
ValidPoints[defaults + 9] = BITAND;

if (FuncParenCount == 0)
{
/*****
If we're outside of an "if" statement
and all paren's are balanced and on
the RHS allow IF PRINT and SET.
*****/
if (NumberOfifs == NumberOfEndifs && ParenCount == 0 && Equation == RHS)
{
TotValPts = defaults + 12;
ValidPoints[defaults + 10] = IF;
ValidPoints[defaults + 11] = PRINT;
ValidPoints[defaults + 12] = SET;
LikelyNextChoice = IF;
}
/*****
If we're in the premise and not defining
function arguments allow AND OR BITXOR
BITAND BITOR.
*****/
else if (WhereAmI == PREMISE)
{
/*****
If we're in the PREMISE and our overall paren
count is balanced and we're on the RHS allow
some logical operands .
*****/
if (ParenCount == 0)
{
if(Equation == RHS)
{
TotValPts = defaults + 13;
ValidPoints[defaults + 10] = AND;
ValidPoints[defaults + 11] = OR;
ValidPoints[defaults + 12] = THEN;
LikelyNextChoice = THEN;
}
/*****
If we also happen to be on the LHS of an Equation
allow some relational stuff too.
*****/
else if (Equation == LHS)
{
TotValPts = defaults + 16;
ValidPoints[defaults + 10] = LT;
ValidPoints[defaults + 11] = GT;
ValidPoints[defaults + 12] = LE;
ValidPoints[defaults + 13] = GE;
ValidPoints[defaults + 14] = NE;
ValidPoints[defaults + 15] = EQ;
LikelyNextChoice = EQ;
}
}
/*****
If we're in the PREMISE and our overall paren
count is unbalanced allow R_PAREN and logical
operands.

```



```

*****
else
{
    TotValPts = defaults + 13;
    ValidPoints[defaults + 10] = AND;
    ValidPoints[defaults + 11] = OR;
    ValidPoints[defaults + 12] = R_PAREN;
    LikelyNextChoice = R_PAREN;
}
}
/*****
We're in the CONSEQUENCE and not defining function
arguments.
*****/
else
{
    /*****
    If we're in the CONSEQUENCE and not defining
    function arguments and we have balanced paren's
    allow ENDIF PRINT SET.
    *****/
    if (ParenCount == 0)
    {
        TotValPts = defaults + 14;
        ValidPoints[defaults + 10] = END_IF;
        ValidPoints[defaults + 11] = PRINT;
        ValidPoints[defaults + 12] = SET;
        ValidPoints[defaults + 13] = IF;
        LikelyNextChoice = END_IF;
        /*****
        If we don't already have an else associated with
        the current THEN allow an ELSE.
        *****/
        if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
        {
            ValidPoints[TotValPts++] = ELSE;
            LikelyNextChoice = ELSE;
        }
    }
    else
    {
        /*****
        If we don't have balanced paren's be better give
        them the opportunity to balance them.
        *****/
        TotValPts = defaults + 11;
        ValidPoints[defaults + 10] = R_PAREN;
        LikelyNextChoice = R_PAREN;
    }
}
}
/*****
We're defining function arguments.
*****/
else
{
    TotValPts = defaults + 11;
    ValidPoints[defaults + 10] = R_PAREN;
    if (FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
        ValidPoints[TotValPts++] = COMMA;
    LikelyNextChoice = R_PAREN;
}
}
/*****
SET
*****/
else if (on_my_left == SET)
{
    TotValPts = defaults + 2;
    ValidPoints[defaults + 0] = SIGNAL;
    ValidPoints[defaults + 1] = LOCAL;
    LikelyNextChoice = SIGNAL;
}
/*****
PRINT
*****/
else if (on_my_left == PRINT)
{

```

```

/*****
If an ELSE has already been used in this if
we don't let them use it again!
*****/
if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    TotValPts = defaults + 4;
else TotValPts = defaults + 4;
ValidPoints[defaults + 0] = IF;
ValidPoints[defaults + 1] = SET;
ValidPoints[defaults + 2] = PRINT;
LikelyNextChoice = PRINT;
if (NumberOfIfs - NumberOfEndifs > 0)
{
    TotValPts += 1;
    ValidPoints[TotValPts - 1] = END_IF;
    LikelyNextChoice = END_IF;
    /****
    If we don't already have an else associated with
    the current THEN allow an ELSE.
    *****/
    if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    {
        TotValPts += 1;
        ValidPoints[TotValPts - 1] = ELSE;
        LikelyNextChoice = ELSE;
        LikelyNextChoice = ELSE;
    }
}
/*****
NEW, DOCUMENT
*****/
else if (on_my_left == NEW || on_my_left == HELP)
{
    TotValPts = defaults + 3;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SET;
    ValidPoints[defaults + 2] = PRINT;
    LikelyNextChoice = IF;
}
/*****
END_IF
*****/
else if (on_my_left == END_IF)
{
    TotValPts = defaults + 3;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SET;
    ValidPoints[defaults + 2] = PRINT;
    /****
    If the difference between the if and endifs
    is 0, then we don't need any more endifs.
    If an ELSE has already been used in this if
    we don't let them use it again!
    *****/
    if ((NumberOfIfs-NumberOfEndifs) > 0)
    {
        ValidPoints[TotValPts++] = END_IF;
        LikelyNextChoice = IF;
    }
    if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    {
        ValidPoints[TotValPts++] = ELSE;
        LikelyNextChoice = ELSE;
    }
}
/*****
DELETE
*****/
if (on_my_right == DELETE)
    LikelyNextChoice = DELETE;
/****
For when we just have started up.
*****/
if (on_my_right == 99)
{
    TotValPts = 10;
    ValidPoints[0] = QUIT;
}

```

90/06/07
16:39:28

next_inputs.c

11

```
ValidPoints[1] = RETRIEVE;  
ValidPoints[2] = REMOVE;  
ValidPoints[3] = BACKUP;  
ValidPoints[4] = SMALL;  
ValidPoints[5] = MEDIUM;  
ValidPoints[6] = LARGE;  
ValidPoints[7] = HELP;  
ValidPoints[8] = LIST;  
ValidPoints[9] = NEW;  
LikelyNextChoice = RETRIEVE;
```

90/06/07
16:42:59

nf.c

1

```
#include "nf.h"
#define MAX_MSGS 20
#define MAX_COMPS 50
#define UNLIKELY_VALUE -999
/*****
Purpose: This file contains noise filtering routines. There
is one noise filter routine for every output routine in the
interface. For an explanation of these output routines see
the user's guide for IESP. There are two different
algorithms used in all the routines discounting types (i.e.
short, float, etc.). The simpler routine handles string
output, since we don't want to store and compare strings.
The more complicated routine handles all numerical data.
*****/
```

Designer: Troy Heindel/NASA, Erick Kindred/Dual & John Muratore/NASA

Prototype: Troy Heindel/NASA using HyperCard

Programmer: Troy Heindel/NASA

Date: 7/11/88

Version: 1.0

Project: INCO Expert System Project

Revised by:

Reasons for Revision:

External Interfaces

```
fltmsg_issue (Class, Text)
fltmsg_issue (Class, Text)
putsig_off_str (Sig_name, Sig_data)
putsig_str (Sig_name, Sig_data)
putsig_c (Sig_name, Sig_data)
putsig_s (Sig_name, Sig_data)
putsig_f (Sig_name, Sig_data)
putsig_d (Sig_name, Sig_data)
putsig_u (Sig_name, Sig_data)
putsig_i (Sig_name, Sig_data)
putsig_l (Sig_name, Sig_data)
putsig_off_c (Sig_name, Sig_data)
putsig_off_s (Sig_name, Sig_data)
putsig_off_f (Sig_name, Sig_data)
putsig_off_d (Sig_name, Sig_data)
putsig_off_u (Sig_name, Sig_data)
putsig_off_i (Sig_name, Sig_data)
putsig_off_l (Sig_name, Sig_data)
```

```
*****/
fltmsg_issue_NF (faultMsg, color, index, noiseFilter, out)
char faultMsg[]; /* The message the user wishes to print */
int color, /* The color number; either 1, 2, or 3 */
index, /* The index to the fault in the comp */
noiseFilter; /* The noise filter value for this comp */
struct faultMsgStruct *out;
{
extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((strcmp (out[index].estFaultMsg, faultMsg) != 0) &&
(out[index].countDownFaultMsg < noiseFilter))
{
if (++out[index].countDownFaultMsg == noiseFilter)
out[index].estFaultMsg[0] = '\0';
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopFaultMsg) > *rate_ptr)
{
out[index].countUpFaultMsg = 0;
out[index].countDownFaultMsg = 0;
/*****
If we have consistently not fired, then invalidate the
previously established value.
*****/
if (abs(iteration_ctr - out[index].loopFaultMsg) > (*rate_ptr * noiseFilter))
out[index].estFaultMsg[0] = '\0';
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopFaultMsg = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (strcmp (out[index].prevFaultMsg, faultMsg) != 0)
{
out[index].countUpFaultMsg = 0;
strcpy (out[index].prevFaultMsg, faultMsg);
}
else
{
/*****
If the count_up is less than noiseFilter then increment it.
*****/
if (out[index].countUpFaultMsg < noiseFilter) /* Stop us from wrap-around */
{
/*****
If the incremented count_up = noiseFilter then give 'em output, and
reset the established value to the newly established value.
*****/
if (++out[index].countUpFaultMsg == noiseFilter)
{
if (strcmp (out[index].estFaultMsg, faultMsg) != 0)
{
strcpy (out[index].estFaultMsg, faultMsg);
fltmsg_issue (color, faultMsg);
}
}
}

/*****
Initialize the invalidation counter to 1.
*****/
out[index].countDownFaultMsg = 0;
}
}
```

30/06/07
16:42:59

nf.c

3

```
/******  
putsig_str_NF (Sig_name, Sig_data, index, noiseFilter, out)  
char Sig_name[], /* The name of the signal put */  
    Sig_data[], /* The data for signal put */  
int index, /* The index to the signal in the comp */  
    noiseFilter; /* The noise filter value for this comp */  
struct sigStringStruct *out;  
  
{  
    extern int *rate_ptr; /* Global pointer to comp's noise filter value */  
    extern int iteration_ctr; /* Global; How many times the group has looped */  
  
    /******  
    Are we deviating from the established value? If we are doing  
    so consistently, then invalidate the established value.  
    *****/  
    if ((strcmp (out[index].estSigString, Sig_data) != 0) &&  
        (out[index].countDownSigString < noiseFilter))  
    {  
        if (++out[index].countDownSigString == noiseFilter)  
        {  
            out[index].estSigString[0] = '\0';  
        }  
    }  
  
    /******  
    If we are not firing continuously, then send everyone back to  
    the starting values.  
    *****/  
    if (abs(iteration_ctr - out[index].loopSigString) > *rate_ptr)  
    {  
        out[index].countUpSigString = 0;  
        out[index].countDownSigString = 0;  
        /******  
        If we have consistently not fired, then invalidate the  
        previously established value.  
        *****/  
        if (abs(iteration_ctr - out[index].loopSigString) > (*rate_ptr * noiseFilter))  
            out[index].estSigString[0] = '\0';  
    }  
  
    /******  
    Reset the loop count_up to the new loop count_up.  
    *****/  
    out[index].loopSigString = iteration_ctr;  
  
    /******  
    If the current value is different from the last value, then  
    start the count_up at 1 and reset the previous value, else...  
    *****/  
    if (strcmp (out[index].prevSigString, Sig_data) != 0)  
    {  
        out[index].countUpSigString = 0;  
        strcpy (out[index].prevSigString, Sig_data);  
    }  
    else  
    {  
        /******  
        If the count_up is less than noiseFilter then increment it.  
        *****/  
        if (out[index].countUpSigString < noiseFilter) /* Stop us from wrap-around */  
        {  
            /******  
            If the incremented count_up = noiseFilter then give 'em output, and  
            reset the established value to the newly established value.  
            *****/  
            if (++out[index].countUpSigString == noiseFilter)  
            {  
                if (strcmp (out[index].estSigString, Sig_data) != 0)  
                {  
                    putsig_str (Sig_name, Sig_data);  
                    strcpy (out[index].estSigString, Sig_data);  
                }  
            }  
            /******  
            Initialize the invalidation counter to 1.  
            *****/  
            out[index].countDownSigString = 0;  
        }  
    }  
}
```

90/06/07
16:42:59

nf.c

5


```

/*****/
putsig_off_str_NF (Sig_name, Sig_data, index, noiseFilter, out)
char Sig_name[], /* The name of the signal put */
  Sig_data[]; /* The data for signal put */
int index, /* The index to the signal in the comp */
  noiseFilter; /* The noise filter value for this comp */
struct sigOffStringStruct *out;

(
extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****/
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
/*****/
if ((strcmp (out[index].estSigOffString, Sig_data) != 0) &&
    (out[index].countDownSigOffString < noiseFilter))
{
    if (++out[index].countDownSigOffString == noiseFilter)
        out[index].estSigOffString[0] = '\0';
}

/*****/
If we are not firing continuously, then send everyone back to
the starting values.
/*****/
if (abs(iteration_ctr - out[index].loopSigOffString) > *rate_ptr)
{
    out[index].countUpSigOffString = 0;
    out[index].countDownSigOffString = 0;
    /*****/
    If we have consistently not fired, then invalidate the
    previously established value.
    /*****/
    if (abs(iteration_ctr - out[index].loopSigOffString) > (*rate_ptr * noiseFilter))
        out[index].estSigOffString[0] = '\0';
}

/*****/
Reset the loop count_up to the new loop count_up.
/*****/
out[index].loopSigOffString = iteration_ctr;

/*****/
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
/*****/
if (strcmp (out[index].prevSigOffString, Sig_data) != 0)
{
    out[index].countUpSigOffString = 0;
    strcpy (out[index].prevSigOffString, Sig_data);
}
else
{
    /*****/
    If the count_up is less than noiseFilter then increment it.
    /*****/
    if (out[index].countUpSigOffString < noiseFilter) /* Stop us from wrap-around */
    {
        /*****/
        If the incremented count_up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        /*****/
        if (++out[index].countUpSigOffString == noiseFilter)
        {
            if (strcmp (out[index].estSigOffString, Sig_data) != 0)
            {
                putsig_off_str (Sig_name, Sig_data);
                strcpy (out[index].estSigOffString, Sig_data);
            }
        }
        /*****/
        Initialize the invalidation counter to 1.
        /*****/
        out[index].countDownSigOffString = 0;
    }
}
)
)

```

90/06/07
16:42:59

nf.c

7

```

/*****
putsig_s_NF (Sig_name, Sig_data, index, noiseFilter, out)
char  Sig_name[]; /* The name of the signal put */
short Sig_data;   /* The data for signal put */
int   index,      /* The index to the signal in the comp */
      noiseFilter; /* The noise filter value for this comp */
struct sigShortStruct *out;

{
    extern int *rate_ptr; /* Global pointer to comp's noise filter value */
    extern int iteration_ctr; /* Global; How many times the group has looped */

    /*****
    Are we deviating from the established value? If we are doing
    so consistently, then invalidate the established value.
    *****/
    if ((out[index].estSigShort != Sig_data) &&
        (out[index].countDownSigShort < noiseFilter))
    {
        if (++out[index].countDownSigShort == noiseFilter)
            out[index].estSigShort = UNLIKELY_VALUE;
    }

    /*****
    If we are not firing continuously, then send everyone back to
    the starting values.
    *****/
    if (abs(iteration_ctr - out[index].loopSigShort) > *rate_ptr)
    {
        out[index].countUpSigShort = 0;
        out[index].countDownSigShort = 0;
        /*****
        If we have consistently not fired, then invalidate the
        previously established value.
        *****/
        if (abs(iteration_ctr - out[index].loopSigShort) > (*rate_ptr * noiseFilter))
            out[index].estSigShort = UNLIKELY_VALUE;
    }

    /*****
    Reset the loop count_up to the new loop count_up.
    *****/
    out[index].loopSigShort = iteration_ctr;

    /*****
    If the current value is different from the last value, then
    start the count_up at 1 and reset the previous value, else...
    *****/
    if (out[index].prevSigShort != Sig_data)
    {
        out[index].countUpSigShort = 0;
        out[index].prevSigShort = Sig_data;
    }
    else
    {
        /*****
        If the count_up is less than noiseFilter then increment it.
        *****/
        if (out[index].countUpSigShort < noiseFilter) /* Stop us from wrap-around */
        {
            /*****
            If the incremented count_up = noiseFilter then give 'em output, and
            reset the established value to the newly established value.
            *****/
            if (++out[index].countUpSigShort == noiseFilter)
            {
                if (out[index].estSigShort != Sig_data)
                {
                    putsig_s (Sig_name, Sig_data);
                    out[index].estSigShort = Sig_data;
                }
            }
            /*****
            Initialize the invalidation counter to 1.
            *****/
            out[index].countDownSigShort = 0;
        }
    }
}

```

90/06/07
16:42:59

nf.c

9

```

/*****
putsig_off_s_NF (Sig_name, Sig_data, index, noiseFilter, out)
char   Sig_name[]; /* The name of the signal put */
short  Sig_data;    /* The data for signal put */
int     index,      /* The index to the signal in the comp */
        noiseFilter; /* The noise filter value for this comp */
struct sigOffShortStruct *out;

{
    extern int *rate_ptr; /* Global pointer to comp's noise filter value */
    extern int iteration_ctr; /* Global; How many times the group has looped */

    /*****
    Are we deviating from the established value? If we are doing
    so consistently, then invalidate the established value.
    *****/
    if ((out[index].estSigOffShort != Sig_data) &&
        (out[index].countDownSigOffShort < noiseFilter))
    {
        if (++out[index].countDownSigOffShort == noiseFilter)
            out[index].estSigOffShort = UNLIKELY_VALUE;
    }

    /*****
    If we are not firing continuously, then send everyone back to
    the starting values.
    *****/
    if (abs(iteration_ctr - out[index].loopSigOffShort) > *rate_ptr)
    {
        out[index].countUpSigOffShort = 0;
        out[index].countDownSigOffShort = 0;
        /*****
        If we have consistently not fired, then invalidate the
        previously established value.
        *****/
        if (abs(iteration_ctr - out[index].loopSigOffShort) > (*rate_ptr * noiseFilter))
            out[index].estSigOffShort = UNLIKELY_VALUE;
    }

    /*****
    Reset the loop count_up to the new loop count_up.
    *****/
    out[index].loopSigOffShort = iteration_ctr;

    /*****
    If the current value is different from the last value, then
    start the count_up at 1 and reset the previous value, else...
    *****/
    if (out[index].prevSigOffShort != Sig_data)
    {
        out[index].countUpSigOffShort = 0;
        out[index].prevSigOffShort = Sig_data;
    }
    else
    {
        /*****
        If the count_up is less than noiseFilter then increment it.
        *****/
        if (out[index].countUpSigOffShort < noiseFilter) /* Stop us from wrap-around */
        {
            /*****
            If the incremented count_up = noiseFilter then give 'em output, and
            reset the established value to the newly established value.
            *****/
            if (++out[index].countUpSigOffShort == noiseFilter)
            {
                if (out[index].estSigOffShort != Sig_data)
                {
                    putsig_s (Sig_name, Sig_data);
                    out[index].estSigOffShort = Sig_data;
                }
            }
            /*****
            Initialize the invalidation counter to 1.
            *****/
            out[index].countDownSigOffShort = 0;
        }
    }
}

```

90/06/07
16:42:59

nf.c

11

```

/*****
putsig_i NF (Sig_name, Sig_data, index, noiseFilter, out)
char Sig_name[]; /* The name of the signal put */
int Sig_data, /* The data for signal put */
index, /* The index to the signal in the comp */
noiseFilter; /* The noise filter value for this comp */
struct sigIntStruct *out;

extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigInt != Sig_data) &&
    (out[index].countDownSigInt < noiseFilter))
{
    if (++out[index].countDownSigInt == noiseFilter)
        out[index].estSigInt = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigInt) > *rate_ptr)
{
    out[index].countUpSigInt = 0;
    out[index].countDownSigInt = 0;
    /*****
    If we have consistently not fired, then invalidate the
    previously established value.
    *****/
    if (abs(iteration_ctr - out[index].loopSigInt) > (*rate_ptr * noiseFilter))
        out[index].estSigInt = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigInt = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigInt != Sig_data)
{
    out[index].countUpSigInt = 0;
    out[index].prevSigInt = Sig_data;
}
else
{
    /*****
    If the count_up is less than noiseFilter then increment it.
    *****/
    if (out[index].countUpSigInt < noiseFilter) /* Stop us from wrap-around */
    {
        /*****
        If the incremented count_up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        *****/
        if (++out[index].countUpSigInt == noiseFilter)
        {
            if (out[index].estSigInt != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigInt = Sig_data;
            }
        }
        /*****
        Initialize the invalidation counter to 1.
        *****/
        out[index].countDownSigInt = 0;
    }
}
}

```

90/06/07
16:42:59

nf.c

13


```

/*****
putsig_off_i_NF (Sig_name, Sig_data, index, noiseFilter, out)
char Sig_name[]; /* The name of the signal put */
int Sig_data, /* The data for signal put */
index, /* The index to the signal in the comp */
noiseFilter; /* The noise filter value for this comp */
struct sigOffIntStruct *out;

{
extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigOffInt != Sig_data) &&
(out[index].countDownSigOffInt < noiseFilter))
{
if (++out[index].countDownSigOffInt == noiseFilter)
out[index].estSigOffInt = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigOffInt) > *rate_ptr)
{
out[index].countUpSigOffInt = 0;
out[index].countDownSigOffInt = 0;
/*****
If we have consistently not fired, then invalidate the
previously established value.
*****/
if (abs(iteration_ctr - out[index].loopSigOffInt) > (*rate_ptr * noiseFilter))
out[index].estSigOffInt = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigOffInt = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigOffInt != Sig_data)
{
out[index].countUpSigOffInt = 0;
out[index].prevSigOffInt = Sig_data;
}
else
{
/*****
If the count_up is less than noiseFilter then increment it.
*****/
if (out[index].countUpSigOffInt < noiseFilter) /* Stop us from wrap-around */
{
/*****
If the incremented count_up = noiseFilter then give 'em output, and
reset the established value to the newly established value.
*****/
if (++out[index].countUpSigOffInt == noiseFilter)
{
if (out[index].estSigOffInt != Sig_data)
{
putsig_s (Sig_name, Sig_data);
out[index].estSigOffInt = Sig_data;
}
}
/*****
Initialize the invalidation counter to 1.
*****/
out[index].countDownSigOffInt = 0;
}
}
}
}

```

30/06/07
16:42:59

nf.c

15

```

/*****
putsig_f_NF (Sig_name, Sig_data, index, noiseFilter, out)
char  Sig_name[]; /* The name of the signal put */
float Sig_data;   /* The data for signal put */
int   index;      /* The index to the signal in the comp */
noiseFilter; /* The noise filter value for this comp */
struct sigFloatStruct *out;

{
extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigFloat != Sig_data) &&
    (out[index].countDownSigFloat < noiseFilter))
{
    if (++out[index].countDownSigFloat == noiseFilter)
        out[index].estSigFloat = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigFloat) > *rate_ptr)
{
    out[index].countUpSigFloat = 0;
    out[index].countDownSigFloat = 0;
    /*****
    If we have consistently not fired, then invalidate the
    previously established value.
    *****/
    if (abs(iteration_ctr - out[index].loopSigFloat) > (*rate_ptr * noiseFilter))
        out[index].estSigFloat = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigFloat = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigFloat != Sig_data)
{
    out[index].countUpSigFloat = 0;
    out[index].prevSigFloat = Sig_data;
}
else
{
    /*****
    If the count_up is less than noiseFilter then increment it.
    *****/
    if (out[index].countUpSigFloat < noiseFilter) /* Stop us from wrap-around */
    {
        /*****
        If the incremented count_up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        *****/
        if (++out[index].countUpSigFloat == noiseFilter)
        {
            if (out[index].estSigFloat != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigFloat = Sig_data;
            }
        }
        /*****
        Initialize the invalidation counter to 1.
        *****/
        out[index].countDownSigFloat = 0;
    }
}
}
}

```

90/06/07
16:42:59

nf.c

17

```

/*****
putsig_off_f_NF (Sig_name, Sig_data, index, noiseFilter, out)
char  Sig_name[]; /* The name of the signal put */
float Sig_data;    /* The data for signal put */
int   index;       /* The index to the signal in the comp */
      noiseFilter; /* The noise filter value for this comp */
struct sigOffFloatStruct *out;

{
extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigOffFloat != Sig_data) &&
    (out[index].countDownSigOffFloat < noiseFilter))
{
    if (++out[index].countDownSigOffFloat == noiseFilter)
        out[index].estSigOffFloat = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigOffFloat) > *rate_ptr)
{
    out[index].countUpSigOffFloat = 0;
    out[index].countDownSigOffFloat = 0;
/*****
If we have consistently not fired, then invalidate the
previously established value.
*****/
    if (abs(iteration_ctr - out[index].loopSigOffFloat) > (*rate_ptr * noiseFilter))
        out[index].estSigOffFloat = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigOffFloat = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigOffFloat != Sig_data)
{
    out[index].countUpSigOffFloat = 0;
    out[index].prevSigOffFloat = Sig_data;
}
else
{
/*****
If the count_up is less than noiseFilter then increment it.
*****/
    if (out[index].countUpSigOffFloat < noiseFilter) /* Stop us from wrap-around */
    {
/*****
If the incremented count_up = noiseFilter then give 'em output, and
reset the established value to the newly established value.
*****/
        if (++out[index].countUpSigOffFloat == noiseFilter)
        {
            if (out[index].estSigOffFloat != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigOffFloat = Sig_data;
            }
        }
/*****
Initialize the invalidation counter to 1.
*****/
        out[index].countDownSigOffFloat = 0;
    }
}
}

```

90/06/07
16:42:59

nf.c

19

```

/*****
putsig_d_NF (Sig_name, Sig_data, index, noiseFilter, out)
char   Sig_name[]; /* The name of the signal put */
double Sig_data; /* The data for signal put */
int    index, /* The index to the signal in the comp */
noiseFilter; /* The noise filter value for this comp */
struct sigDoubleStruct *out;

extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigDouble != Sig_data) &&
    (out[index].countDownSigDouble < noiseFilter))
{
    if (++out[index].countDownSigDouble == noiseFilter)
        out[index].estSigDouble = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigDouble) > *rate_ptr)
{
    out[index].countUpSigDouble = 0;
    out[index].countDownSigDouble = 0;
    /*****
    If we have consistently not fired, then invalidate the
    previously established value.
    *****/
    if (abs(iteration_ctr - out[index].loopSigDouble) > (*rate_ptr * noiseFilter))
        out[index].estSigDouble = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigDouble = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigDouble != Sig_data)
{
    out[index].countUpSigDouble = 0;
    out[index].prevSigDouble = Sig_data;
}
else
{
    /*****
    If the count_up is less than noiseFilter then increment it.
    *****/
    if (out[index].countUpSigDouble < noiseFilter) /* Stop us from wrap-around */
    {
        /*****
        If the incremented count_up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        *****/
        if (++out[index].countUpSigDouble == noiseFilter)
        {
            if (out[index].estSigDouble != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigDouble = Sig_data;
            }
        }
        /*****
        Initialize the invalidation counter to 1.
        *****/
        out[index].countDownSigDouble = 0;
    }
}
}

```

90/06/07
16:42:59

nf.c

21


```

/*****
putsig_off_d_NF (Sig_name, Sig_data, index, noiseFilter, out)
char   Sig_name[];      /* The name of the signal put */
double Sig_data;        /* The data for signal put */
int     index,           /* The index to the signal in the comp */
noiseFilter; /* The noise filter value for this comp */
struct sigOffDoubleStruct *out;

extern int *rate_ptr;      /* Global pointer to comp's noise filter value */
extern int iteration_ctr;  /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigOffDouble != Sig_data) &&
    (out[index].countDownSigOffDouble < noiseFilter))
{
    if (++out[index].countDownSigOffDouble == noiseFilter)
        out[index].estSigOffDouble = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigOffDouble) > *rate_ptr)
{
    out[index].countUpSigOffDouble = 0;
    out[index].countDownSigOffDouble = 0;
    /*****
    If we have consistently not fired, then invalidate the
    previously established value.
    *****/
    if (abs(iteration_ctr - out[index].loopSigOffDouble) > (*rate_ptr * noiseFilter))
        out[index].estSigOffDouble = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigOffDouble = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigOffDouble != Sig_data)
{
    out[index].countUpSigOffDouble = 0;
    out[index].prevSigOffDouble = Sig_data;
}
else
{
    /*****
    If the count_up is less than noiseFilter then increment it.
    *****/
    if (out[index].countUpSigOffDouble < noiseFilter) /* Stop us from wrap-around */
    {
        /*****
        If the incremented count_up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        *****/
        if (++out[index].countUpSigOffDouble == noiseFilter)
        {
            if (out[index].estSigOffDouble != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigOffDouble = Sig_data;
            }
        }
        /*****
        Initialize the invalidation counter to 1.
        *****/
        out[index].countDownSigOffDouble = 0;
    }
}
}

```

90/06/07
10:42:59

nf.c

23

```

/*****
putsig_u_NF (Sig_name, Sig_data, index, noiseFilter, out)
char    Sig_name[]; /* The name of the signal put */
unsigned Sig_data;   /* The data for signal put */
int     index,       /* The index to the signal in the comp */
        noiseFilter; /* The noise filter value for this comp */
struct sigUnsignedStruct *out;

extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigUnsigned != Sig_data) &&
    (out[index].countDownSigUnsigned < noiseFilter))
{
    if (++out[index].countDownSigUnsigned == noiseFilter)
        out[index].estSigUnsigned = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigUnsigned) > *rate_ptr)
{
    out[index].countUpSigUnsigned = 0;
    out[index].countDownSigUnsigned = 0;
    /*****
    If we have consistently not fired, then invalidate the
    previously established value.
    *****/
    if (abs(iteration_ctr - out[index].loopSigUnsigned) > (*rate_ptr * noiseFilter))
        out[index].estSigUnsigned = UNLIKELY_VALUE;
}

/*****
Reset the loop count up to the new loop count up.
*****/
out[index].loopSigUnsigned = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigUnsigned != Sig_data)
{
    out[index].countUpSigUnsigned = 0;
    out[index].prevSigUnsigned = Sig_data;
}
else
{
    /*****
    If the count up is less than noiseFilter then increment it.
    *****/
    if (out[index].countUpSigUnsigned < noiseFilter) /* Stop us from wrap-around */
    {
        /*****
        If the incremented count up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        *****/
        if (++out[index].countUpSigUnsigned == noiseFilter)
        {
            if (out[index].estSigUnsigned != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigUnsigned = Sig_data;
            }
        }
        /*****
        Initialize the invalidation counter to 1.
        *****/
        out[index].countDownSigUnsigned = 0;
    }
}
}

```

90/06/07
16:42:59

nf.c

25

```

/*****
putsig_off_u_NF (Sig_name, Sig_data, index, noiseFilter, out)
char    Sig_name[];    /* The name of the signal put */
unsigned Sig_data;    /* The data for signal put */
int     index,        /* The index to the signal in the comp */
noiseFilter; /* The noise filter value for this comp */
struct sigOffUnsignedStruct *out;

{
    extern int *rate_ptr;    /* Global pointer to comp's noise filter value */
    extern int iteration_ctr; /* Global; How many times the group has looped */

    /*****
    Are we deviating from the established value? If we are doing
    so consistently, then invalidate the established value.
    *****/
    if ((out[index].estSigOffUnsigned != Sig_data) &&
        (out[index].countDownSigOffUnsigned < noiseFilter))
    {
        if (++out[index].countDownSigOffUnsigned == noiseFilter)
            out[index].estSigOffUnsigned = UNLIKELY_VALUE;
    }

    /*****
    If we are not firing continuously, then send everyone back to
    the starting values.
    *****/
    if (abs(iteration_ctr - out[index].loopSigOffUnsigned) > *rate_ptr)
    {
        out[index].countUpSigOffUnsigned = 0;
        out[index].countDownSigOffUnsigned = 0;
        /*****
        If we have consistently not fired, then invalidate the
        previously established value.
        *****/
        if (abs(iteration_ctr - out[index].loopSigOffUnsigned) > (*rate_ptr * noiseFilter))
            out[index].estSigOffUnsigned = UNLIKELY_VALUE;
    }

    /*****
    Reset the loop count_up to the new loop count_up.
    *****/
    out[index].loopSigOffUnsigned = iteration_ctr;

    /*****
    If the current value is different from the last value, then
    start the count_up at 1 and reset the previous value, else...
    *****/
    if (out[index].prevSigOffUnsigned != Sig_data)
    {
        out[index].countUpSigOffUnsigned = 0;
        out[index].prevSigOffUnsigned = Sig_data;
    }
    else
    {
        /*****
        If the count_up is less than noiseFilter then increment it.
        *****/
        if (out[index].countUpSigOffUnsigned < noiseFilter) /* Stop us from wrap-around */
        {
            /*****
            If the incremented count_up = noiseFilter then give 'em output, and
            reset the established value to the newly established value.
            *****/
            if (++out[index].countUpSigOffUnsigned == noiseFilter)
            {
                if (out[index].estSigOffUnsigned != Sig_data)
                {
                    putsig_s (Sig_name, Sig_data);
                    out[index].estSigOffUnsigned = Sig_data;
                }
            }
            /*****
            Initialize the invalidation counter to 1.
            *****/
            out[index].countDownSigOffUnsigned = 0;
        }
    }
}

```

900607
1042559

nf.c

27

```

/*****
putsig_l_NF (Sig_name, Sig_data, index, noiseFilter, out)
char   Sig_name[]; /* The name of the signal put */
long   Sig_data;   /* The data for signal put */
int     index,     /* The index to the signal in the comp */
        noiseFilter; /* The noise filter value for this comp */
struct sigLongStruct *out;

(
extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigLong != Sig_data) &&
    (out[index].countDownSigLong < noiseFilter))
{
    if (++out[index].countDownSigLong == noiseFilter)
        out[index].estSigLong = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigLong) > *rate_ptr)
{
    out[index].countUpSigLong = 0;
    out[index].countDownSigLong = 0;
    /*****
    If we have consistently not fired, then invalidate the
    previously established value.
    *****/
    if (abs(iteration_ctr - out[index].loopSigLong) > (*rate_ptr * noiseFilter))
        out[index].estSigLong = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigLong = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigLong != Sig_data)
{
    out[index].countUpSigLong = 0;
    out[index].prevSigLong = Sig_data;
}
else
{
    /*****
    If the count_up is less than noiseFilter then increment it.
    *****/
    if (out[index].countUpSigLong < noiseFilter) /* Stop us from wrap-around */
    {
        /*****
        If the incremented count_up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        *****/
        if (++out[index].countUpSigLong == noiseFilter)
        {
            if (out[index].estSigLong != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigLong = Sig_data;
            }
        }
        /*****
        Initialize the invalidation counter to 1.
        *****/
        out[index].countDownSigLong = 0;
    }
}
)

```

90/06/07
16:42:59

nf.c

29


```

/*****
putsig_off_1_NF (Sig_name, Sig_data, index, noiseFilter, out)
char    Sig_name[]; /* The name of the signal put */
long    Sig_data; /* The data for signal put */
int     index, /* The index to the signal in the comp */
noiseFilter; /* The noise filter value for this comp */
struct sigOffLongStruct *out;

{
extern int *rate_ptr; /* Global pointer to comp's noise filter value */
extern int iteration_ctr; /* Global; How many times the group has looped */

/*****
Are we deviating from the established value? If we are doing
so consistently, then invalidate the established value.
*****/
if ((out[index].estSigOffLong != Sig_data) &&
    (out[index].countDownSigOffLong < noiseFilter))
{
    if (++out[index].countDownSigOffLong == noiseFilter)
        out[index].estSigOffLong = UNLIKELY_VALUE;
}

/*****
If we are not firing continuously, then send everyone back to
the starting values.
*****/
if (abs(iteration_ctr - out[index].loopSigOffLong) > *rate_ptr)
{
    out[index].countUpSigOffLong = 0;
    out[index].countDownSigOffLong = 0;
    /*****
    If we have consistently not fired, then invalidate the
    previously established value.
    *****/
    if (abs(iteration_ctr - out[index].loopSigOffLong) > (*rate_ptr * noiseFilter))
        out[index].estSigOffLong = UNLIKELY_VALUE;
}

/*****
Reset the loop count_up to the new loop count_up.
*****/
out[index].loopSigOffLong = iteration_ctr;

/*****
If the current value is different from the last value, then
start the count_up at 1 and reset the previous value, else...
*****/
if (out[index].prevSigOffLong != Sig_data)
{
    out[index].countUpSigOffLong = 0;
    out[index].prevSigOffLong = Sig_data;
}
else
{
    /*****
    If the count_up is less than noiseFilter then increment it.
    *****/
    if (out[index].countUpSigOffLong < noiseFilter) /* Stop us from wrap-around */
    {
        /*****
        If the incremented count_up = noiseFilter then give 'em output, and
        reset the established value to the newly established value.
        *****/
        if (++out[index].countUpSigOffLong == noiseFilter)
        {
            if (out[index].estSigOffLong != Sig_data)
            {
                putsig_s (Sig_name, Sig_data);
                out[index].estSigOffLong = Sig_data;
            }
        }
        /*****
        Initialize the invalidation counter to 1.
        *****/
        out[index].countDownSigOffLong = 0;
    }
}
}
}

```

90/06/07
16:48:34

nf.h

1

```
#define MAX_SIG_NAME_LEN 32
#define MAX_STR_LEN 125

struct faultMsgStruct
{
    char    prevFaultMsg[MAX_STR_LEN];
    char    estFaultMsg[MAX_STR_LEN];
    int     countUpFaultMsg;
    int     countDownFaultMsg;
    int     loopFaultMsg;
};

struct sigStringStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    char    estSigString[MAX_STR_LEN];
    char    prevSigString[MAX_STR_LEN];
    int     countUpSigString;
    int     countDownSigString;
    int     loopSigString;
};

struct sigOffStringStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    char    estSigOffString[MAX_STR_LEN];
    char    prevSigOffString[MAX_STR_LEN];
    int     countUpSigOffString;
    int     countDownSigOffString;
    int     loopSigOffString;
};

struct sigShortStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    short   estSigShort;
    short   prevSigShort;
    short   countUpSigShort;
    short   countDownSigShort;
    short   loopSigShort;
};

struct sigOffShortStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    short   estSigOffShort;
    short   prevSigOffShort;
    short   countUpSigOffShort;
    short   countDownSigOffShort;
    short   loopSigOffShort;
};

struct sigIntStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    int     estSigInt;
    int     prevSigInt;
    int     countUpSigInt;
    int     countDownSigInt;
    int     loopSigInt;
};

struct sigOffIntStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    int     estSigOffInt;
    int     prevSigOffInt;
    int     countUpSigOffInt;
    int     countDownSigOffInt;
    int     loopSigOffInt;
};

struct sigFloatStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    float   estSigFloat;
    float   prevSigFloat;
    float   countUpSigFloat;
}
```

```
float    countDownSigFloat;
float    loopSigFloat;
};

struct sigOffFloatStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    float    estSigOffFloat;
    float    prevSigOffFloat;
    float    countUpSigOffFloat;
    float    countDownSigOffFloat;
    float    loopSigOffFloat;
};

struct sigDoubleStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    double   estSigDouble;
    double   prevSigDouble;
    double   countUpSigDouble;
    double   countDownSigDouble;
    double   loopSigDouble;
};

struct sigOffDoubleStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    double   estSigOffDouble;
    double   prevSigOffDouble;
    double   countUpSigOffDouble;
    double   countDownSigOffDouble;
    double   loopSigOffDouble;
};

struct sigUnsignedStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    unsigned estSigUnsigned;
    unsigned prevSigUnsigned;
    unsigned countUpSigUnsigned;
    unsigned countDownSigUnsigned;
    unsigned loopSigUnsigned;
};

struct sigOffUnsignedStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    unsigned estSigOffUnsigned;
    unsigned prevSigOffUnsigned;
    unsigned countUpSigOffUnsigned;
    unsigned countDownSigOffUnsigned;
    unsigned loopSigOffUnsigned;
};

struct sigLongStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    long    estSigLong;
    long    prevSigLong;
    long    countUpSigLong;
    long    countDownSigLong;
    long    loopSigLong;
};

struct sigOffLongStruct
{
    char    sigName[MAX_SIG_NAME_LEN];
    long    estSigOffLong;
    long    prevSigOffLong;
    long    countUpSigOffLong;
    long    countDownSigOffLong;
    long    loopSigOffLong;
};
```

90/06/07
16:48:41

put_local.c

1

```
/******
```

PUT_LOCAL

Purpose: Put_local handles the local variables.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/27/87

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by: Troy A. Heindel -- 10-27-88

Reasons for Revision:

External Interfaces

```
*****/
```

```
/******
```

Include files

```
*****/
```

```
#include "code.h"
```

```
#include "color.h"
```

```
put_local ()
```

```
{
```

```
    int    rc,          /* The returned values from var_check() */
          localIndex,   /* The index to the variable in the CompVars struct */
          globalIndex;  /* This index is not of use to local variables */
```

```
    char    local[MAX_VAR_LEN], /* String for temporary variable storage */
           msgStr[150];
```

```
    /*
```

```
     * Get the name the local from the user
```

```
     */
```

```
    if (getVarName (local, LOCAL) == ERROR)
        return ERROR;
```

```
    /*
```

```
     * Now check to see if the name is proper
```

```
     */
```

```
    if (var_check (local, &localIndex, &globalIndex, "local") == ERROR)
```

```
    {
```

```
        sprintf(msgStr, "%s has been used previously as a Signal variable", local);
```

```
        inform(msgStr, GREEN, 2);
```

```
        ask("Signal and Local variables can not share names.\nHit [RETURN] to continue", RED, msgStr);
```

```
        return ERROR;
```

```
    }
```

```
    /*
```

```
     * Check to see that the type comparison is correct.
```

```
     * If we are on the right hand side of an Equation,
```

```
     * the local is defined, and the types aren't kosher.
```

```
     */
```

```
    if (Equation == RHS && localIndex != -1 &&
        type_check(CompVars[localIndex].type[0]) == ERROR)
```

```
    {
```

```
        sprintf (msgStr, "%s is of wrong type. Should have been of type '%s' -- Hit [RETURN]", local, CompareType[NumberOfCompares-1]);
```

```
        ask (msgStr, GREEN, msgStr);
```

```
        return ERROR;
```

```
    }
```

```
    /******
```

```
     Clean up indentation and add msid to comp
```

```
    /******/
```

```
    indent (PREMISE);
```

```
    strcat (Comp, local);
```

```
    /******
```

```
     Place the token choice into the history of
     token choices in case of delete.
```

```

*****/
PrevChoice[ChoiceCounter++] = LOCAL;
/*****
If the local variable was not previously used
in this comp it will have a local index = -1.
If this is the case we need to get type info.
*****/
if (localIndex != -1)
{
    ++CompVars[localIndex].occurrence;
    /*****
    if (LHS) then set the CompareType to type of this local
    *****/
    if (Equation == LHS)
        strcpy (CompareType[NumberOfCompares++], CompVars[localIndex].type);
}
else /* localIndex == -1 */
{
    CompVars[NumCompVars].lo1_limit = 0;
    CompVars[NumCompVars].lo2_limit = 25;
    CompVars[NumCompVars].hi1_limit = 50;
    CompVars[NumCompVars].hi2_limit = 100;
    CompVars[NumCompVars].put_or_get = GET;
    strcpy (CompVars[NumCompVars].name, local);
    strcpy (CompVars[NumCompVars].class, "local");
    CompVars[NumCompVars].occurrence = 1;
    sprintf(CompVars[NumCompVars].nomenclature, "/* %s */",
        CompVars[NumCompVars].name);
    /*****
    If LHS then we get the type set the CompareType to it
    and increment the NumberOfCompares counter
    *****/
    if (Equation == LHS)
    {
        get_type(CompVars[NumCompVars].type, "Of what type?");
        strcpy (CompareType[NumberOfCompares++], CompVars[NumCompVars].type);
    }
    /*****
    If RHS then we default the local to the CompareType
    *****/
    else
    {
        strcpy (CompVars[NumCompVars].type, CompareType[NumberOfCompares-1]);
    }
    NumCompVars++; /* increment the count of comp variables */
} /* end of new local definition */
NeedToSave = TRUE;
}

getVarName(nameToGet, class)
char *nameToGet;
int class; /* Either LOCAL, MSID, or SIGNAL */
{
    int rc;

    /*****
    Keep getting name until we get a valid one
    *****/
    do
    {
        if ((rc = ask("Enter the variable name, or (Q)uit this", GREEN, nameToGet)) != ERROR)
        {
            upper (nameToGet);
            /*
            * Now that we have a variable, make sure it doesn't
            * contain unwanted characters.
            */
            rc = goodVar(nameToGet);
        }
        /*
        * First check to see if they just want to leave
        */
        if (nameToGet[0] == 'Q' || rc == DEFAULT)
            return ERROR;
    } while (rc == ERROR);

    return OK;
}

```

20/06/07
16:48:41

put_local.c

3

```
    }
    goodVar(varName)
    char varName[];
    {
        int i; /* Simple Counter */
        char reply[5]; /* String for response from "ask" */

        /*
         * Never allow the first character to be
         * a number.
         */
        if(varName[0] >= '0' && varName[0] <= '9')
        {
            ask("Your input string contained an illegal character -- Hit [RETURN]",GREEN,reply);
            return ERROR;
        }
        /*
         * Does the string contain unwanted chars?
         * Allow only alphaNumerics and underscore.
         */
        for (i=0;i<strlen(varName);i++)
        {
            if (((varName[i] >= 'A' && varName[i] <= 'Z') ||
                (varName[i] >= 'a' && varName[i] <= 'z') ||
                (varName[i] >= '0' && varName[i] <= '9') ||
                varName[i] == '_'))
            {
                ask("Your input string contained an illegal character -- Hit [RETURN]",GREEN,reply);
                return ERROR;
            }
        }
        return OK;
    }
}
```

90/06/07
16:48:55

put_msid.c

1

PUT_MSID

Purpose: Put_msid validates and adds the entered
to the comp string and resets all the
necessary flags.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 6/20/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****
/*****
    Include files
*****
#include "code.h"
#include "color.h"

put_msid ()
{
    int    i,          /* A simple counter */
    local_index,      /* The index to the variable in the CompVars struct */
    global_index;     /* The index to the variable in the MSIDTable */

    char    msid[MSID_NAME_LEN], /* String used to store the name of the msid */
    type[TYPE_LEN], /* A place to put type of the msid */
    nomenclature[NOMEN_LEN], /* A place to put the msid's nomenclature */
    reply[50],
    message[150]; /* String used for displaying message to the user */

    /*
     * Get the name the msid from the user
     */
    if (getVarName (msid, MSID) == ERROR)
        return ERROR;

    /*
     * Try and get an index for the msid
     */
    var_check (msid, &local_index, &global_index, "msid");

    /*****
     * If it's defined locally, show them the nomenclature and
     * increment the occurrence count.
     *****/
    if (local_index != ERROR)
    {
        /*
         * Show 'em the nomenclature for a second.
         */
        inform(CompVars[local_index].nomenclature, GREEN, 1);
        /*****
         * Set up type in case we're on the LHS so we can set the
         * compare type.
         *****/
        strcpy(type, CompVars[local_index].type);
        if (Equation == RHS && type_check(type[0]) == ERROR)
        {
            sprintf (message, "%s is of wrong type. Should have been of type '%s' -- Hit [RETURN]", msid, Co
mpareType[NumberOfCompares-1]);
            ask(message, RED, reply);
            return ERROR;
        }
    }
    else
        CompVars[local_index].occurrence++;
}
```

```

}
/*****
If it's defined globally and not defined locally show them the
nomenclature define it locally using the global information.
*****/
else if (global_index != ERROR && local_index == ERROR)
{
    /*
    * Show 'em the nomenclature for a second.
    */
    inform(MSIDTable[global_index].nomenclature, GREEN, 1);
    strcpy(type, MSIDTable[global_index].type);
    /*****
    Check to see that the type comparison is correct
    *****/
    if (Equation == RHS && type_check(type[0]) == ERROR)
    {
        sprintf (message, "%s is of wrong type. Should have been of type '%s' -- Hit [RETURN]", msid, Co
mpareType[NumberOfCompares-1]);
        ask(message, RED, reply);
        return ERROR;
    }
    else
    {
        strcpy(CompVars[NumCompVars].name, msid);
        strcpy(CompVars[NumCompVars].type, type);
        strcpy(CompVars[NumCompVars].nomenclature, MSIDTable[global_index].nomenclature);
        strcpy(CompVars[NumCompVars].class, "msid");
        CompVars[NumCompVars].occurrence = 1;
        CompVars[NumCompVars].put_or_get = GET;
        CompVars[NumCompVars].lol_limit = 0;
        CompVars[NumCompVars].lo2_limit = 25;
        CompVars[NumCompVars].hil_limit = 50;
        CompVars[NumCompVars].hi2_limit = 100;
        NumCompVars++;
    }
}
/*****
If it's not defined globally - and not defined locally warn them
and get the type and nomenclature info and define it locally.
*****/
else if (global_index == ERROR && local_index == ERROR)
{
    sprintf(message, "Warning: %s is not a defined telemetry point on this system", msid);
    inform(message, PURPLE, 2);
    get_type(type, "Of what type?");
    inform(type, GREEN, 0);
    /*****
    Check to see that the type comparison is correct
    *****/
    if (Equation == RHS && type_check(type[0]) == ERROR)
    {
        sprintf (message, "%s is of wrong type. Should have been of type '%s' -- Hit [RETURN]", msid, Co
mpareType[NumberOfCompares-1]);
        ask(message, RED, reply);
        return ERROR;
    }
    else
    {
        strcpy(CompVars[NumCompVars].name, msid);
        strcpy(CompVars[NumCompVars].type, type);
        ask("Please enter msid nomenclature", GREEN, reply);
        sprintf(nomenclature, " /* %s */", reply);
        strcpy(CompVars[NumCompVars].nomenclature, nomenclature);
        strcpy(CompVars[NumCompVars].class, "msid");
        CompVars[NumCompVars].occurrence = 1;
        CompVars[NumCompVars].put_or_get = GET;
        CompVars[NumCompVars].lol_limit = 0;
        CompVars[NumCompVars].lo2_limit = 25;
        CompVars[NumCompVars].hil_limit = 50;
        CompVars[NumCompVars].hi2_limit = 100;
        NumCompVars++;
    }
}
}

/*****

```


90/06/07
16:48:55

put_msid.c

3

```
if (LHS) we are setting the type for CompareType
*****/
if (Equation == LHS)
    strcpy(CompareType[NumberOfCompares++], type);

/*****
Clean up indentation and add msid to comp
*****/
indent (PREMISE);
strcat (Comp, msid);
PrevChoice[ChoiceCounter++] = MSID;
NeedToSave = TRUE;
}
```

90/06/07
16:49:12

put_signal.c

1

PUT_SIGNAL

Purpose: Put_signal handles the addition of signals into the comp, this includes setting flags and arrays.

Designer: Terri Murphy

Programmer: Terri Murphy

Date: 10/88

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

*****/

Include files

*****/

#include "code.h"

#include "color.h"

put_signal ()

```
{
    char    signal[SIGNAL_NAME_LEN], /* A string to get the signal into */
            nomenclature[NOMEN_LEN], /* A string to get the nomenclature into */
            message[155], /* A place to put your messages */
            reply[50],
            file_name[PATH_LEN]; /* Used to build the file name for the sig tbl */

    int     i, /* A counter */
            rv1, rv2, /* Return values for var_check() */
            is_a_getter, /* TRUE if we are just looking at the signal */
            var_check(), /* Returns index of inputted signal */
            local_index, /* local index to be set by var_check() */
            global_index; /* global index to be set by var_check() */

    /*****
    We need to know if we are putting or getting to
    determine if the signal is valid.
    *****/
    if ((WhereAmI == CONSEQUENCE || (NumberOfIfs - NumberOfEndifs) == 0) && (Equation == LHS))
        is_a_getter = FALSE;
    else
        is_a_getter = TRUE;

    /*****
    Keep getting variables until we get a valid one
    *****/
    do
    {
        /*
        * Get the name the signal from the user
        */
        if (getVarName (signal, SIGNAL) == ERROR)
            return ERROR;

        if ((rv2 = var_check (signal, &local_index, &global_index, "signal")) == ERROR)
        {
            sprintf(message, "%s has been previously defined as a LOCAL variable", signal);
            ask(message, RED, reply);
        }

        /*****
        If the signal is in the signal table we should show the
        nomenclature.
        *****/
        if (global_index != -1)
            inform(SignalTable[global_index].nomenclature, GREEN, 1);
    }
}
```

```

Check to see that the type comparison is correct
*****
if (Equation == RHS && global_index != -1 && type_check(SignalTable[global_index].type[0]) == ERROR)
(
    sprintf (message, "%s is of wrong type. Should have been of type '%s' -- Hit [RETURN]", signal, Sign
alTable[global_index].type);
    ask(message, RED, reply);
    rv2 = ERROR;
)
if (Equation == RHS && local_index != -1 && type_check(CompVars[local_index].type[0]) == ERROR)
(
    sprintf (message, "%s is of wrong type. Should have been of type '%s' -- Hit [RETURN]", signal, Sign
alTable[global_index].type);
    ask(message, RED, reply);
    rv2 = ERROR;
)
} while (rv1 == ERROR || rv2 == ERROR);
/*****
rv2 = 0
local_index = -1
global_index = <valid_index>
This means that the signal already exists in the signal table, but it
hasn't been defined locally yet. Create the CompVars entry.
*****/
if (local_index == -1 && global_index != -1)
(
    strcpy(CompVars[NumCompVars].name, signal);
    CompVars[NumCompVars].occurrence = 1;
    strcpy(CompVars[NumCompVars].class, "signal");
    if (is_a_getter)
        CompVars[NumCompVars].put_or_get = GET;
    else
        CompVars[NumCompVars].put_or_get = PUT;
    CompVars[NumCompVars].lo1_limit = 0;
    CompVars[NumCompVars].lo2_limit = 25;
    CompVars[NumCompVars].hi1_limit = 50;
    CompVars[NumCompVars].hi2_limit = 100;
    strcpy(CompVars[NumCompVars].nomenclature,
        SignalTable[global_index].nomenclature);
    strcpy(CompVars[NumCompVars].type, SignalTable[global_index].type);

    /*****
    if (LHS) we are setting the type for this compare
    *****/
    if (Equation == LHS)
        strcpy (CompareType[NumberOfCompares++], SignalTable[global_index].type);
    NumCompVars++;
)
/*****
rv2 = 0
local_index != -1
Means that the signal is already defined locally.
Update the occurrence count, make sure the
the signal is still being used in the same way.
*****/
else if (local_index != -1)
(
    /*****
    Let's make sure that if they were getting they're
    still getting - or if they're putting now we need
    to change put_or_get to "PET" (and vice versa).
    *****/
    if ((is_a_getter && CompVars[local_index].put_or_get == PUT) ||
        (is_a_getter && CompVars[local_index].put_or_get == GET))
        CompVars[local_index].put_or_get = PET;
    /*****
    Increment the occurrence of this signal
    *****/
    CompVars[local_index].occurrence++;
    /*****
    if (LHS) we are setting the type for this compare
    *****/
    if (Equation == LHS)
        strcpy (CompareType[NumberOfCompares++], CompVars[local_index].type);
)
/*****
If the rv2 is 0
local_index is -1

```

```

global_index is -1
The signal is not defined anywhere - SO DEFINE IT!!!
*****/
else if (local_index == -1 && global_index == -1)
{
    strcpy(CompVars[NumCompVars].name , signal);
    CompVars[NumCompVars].occurrence = 1;
    strcpy(CompVars[NumCompVars].class, "signal");
    CompVars[NumCompVars].put_or_get = PUT;
    CompVars[NumCompVars].lo1_limit = 0;
    CompVars[NumCompVars].lo2_limit = 25;
    CompVars[NumCompVars].hi1_limit = 50;
    CompVars[NumCompVars].hi2_limit = 100;
    ask("Please enter signal nomenclature.", GREEN, nomenclature);
    sprintf(CompVars[NumCompVars].nomenclature, " /* %s */", nomenclature);
    /*****
    If LHS then we get the type set the CompareType to it
    and increment the NumberOfCompares counter
    *****/
    if (Equation == LHS)
    {
        get_type(CompVars[NumCompVars].type, "Of what type?");
        strcpy (CompareType[NumberOfCompares++], CompVars[NumCompVars].type);
    }
    /*****
    If RHS then we default the signal to the CompareType
    *****/
    else
    {
        strcpy (CompVars[NumCompVars].type, CompareType[NumberOfCompares-1]);
    }

    NumCompVars++;
}
/*****
Clean up indentation and add signal to comp
*****/
indent (PREMISE);
strcat (Comp, signal);
PrevChoice[ChoiceCounter++] = SIGNAL;
NeedToSave = TRUE;
}

```

90/06/07
16:52:45

put_status.c

1

PUT_STATUS

Purpose: Put_status places the development status of the comp whether incomplete, complete, installed, or error in the upper left hand corner of the Work Area.

Designer: Terri Murphy

Programmer: Terri Murphy

Date: 4/5/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

*****/

/*****

Include files

*****/

#include "code.h"

#include "color.h"

put_status ()

```
{
    char    status[15], /* The status string */
           response[5]; /* A place to put user input */

    /*****
    The comp is complete if the length of the comp is greater
    than 2 and there are an equal number of "if"s and "endif"s.
    *****/
    if (CompInfo[CompNumber].disposition == ERROR && !NeedToSave)
    {
        strcpy (status, "Error");
    }
    else if (CompInfo[CompNumber].disposition == INSTALLED && !NeedToSave)
    {
        strcpy (status, "Installed");
    }
    else if (NumberOfIfs-NumberOfEndifs == 0)
    {
        if (PrevChoice[ChoiceCounter-1] == END_IF ||
            PrevChoice[ChoiceCounter-1] == PRINT ||
            PrevChoice[ChoiceCounter-1] == COMMENT ||
            Equation == RHS &&
            (PrevChoice[ChoiceCounter-1] == MSID ||
             PrevChoice[ChoiceCounter-1] == NUMBER ||
             PrevChoice[ChoiceCounter-1] == STRING ||
             PrevChoice[ChoiceCounter-1] == SIGNAL ||
             PrevChoice[ChoiceCounter-1] == LOCAL ||
             PrevChoice[ChoiceCounter-1] == PI ||
             (PrevChoice[ChoiceCounter-1] == R_PAREN && ParenCount == 0)))
        {
            strcpy(status, "Complete");
            CompInfo[CompNumber].disposition = COMPLETE;
        }
        else
        {
            strcpy(status, "Incomplete");
            CompInfo[CompNumber].disposition = INCOMPLETE;
        }
    }
    else
    {
        CompInfo[CompNumber].disposition = INCOMPLETE;
        strcpy(status, "Incomplete");
    }
}
```

00/06/07
16:52:45

put_status.c

2

```
/*  
*****  
Put up the status bubble  
*****  
mgihue (BLACK);  
mgrbox (0.1682, 0.9120, 0.35, 0.9320);  
mgihue (BLUE);  
mgrbox (0.1682, 0.9120, 0.1682 + strlen(status)*0.01, 0.9320);  
mgrfc (0.1682, 0.9220, 0.0080);  
mgrfc (0.1682+strlen(status)*0.01, 0.9220, 0.0080);  
mgihue (YELLOW);  
mgrgfs (0.1682,0.9110,0,status);  
  
/*  
 * If there is danger of the comp exceeding the MAX_COMP_LEN if  
 * the largest token (MAX_COMMENT_LEN) is selected next, tell  
 * the user to notify developers and EXIT before irrevocable  
 * damage is done.  
 */  
if(strlen(Comp) > MAX_COMP_LEN - MAX_COMMENT_LEN)  
{  
    ask("You have exceeded the maximum comp length. Notify developers.\nHit [Return] to exit",RED,status);  
    save();  
    cleanExit();  
}
```

90/06/07
16:52:58

put_token.c

1

PUT_ADD

Purpose: Put_add adds the token '+' to the comp string
and resets all the necessary flags.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/6/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

*****/

/*****

Include files

*****/

#include "code.h"

#include <ctype.h>

put_add ()

```
{
    strcat (Comp, "+");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = ADD;
}
```

put_and ()

```
{
    int spaces, i;

    strcat (Comp, " and");
    PrevChoice[ChoiceCounter++] = AND;
    Equation = LHS;
    NeedToSave = TRUE;
}
```

*****/

put_divide ()

```
{
    strcat (Comp, "/");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = DIVIDE;
}
```

*****/

put_else ()

```
{
    int i; /* a counter */

    NestedElseCheck[NumberOfIfs-NumberOfEndifs] = ELSE;
    /*****
    Clean up indentation
    *****/
    strcat (Comp, "\n");
    for (i=0; i<((NumberOfIfs-NumberOfEndifs)-1)*SIZE_INDENT; i++)
        strcat (Comp, " ");
    strcat (Comp, "else");
    PrevChoice[ChoiceCounter++] = ELSE;
    Equation = LHS;
    NeedToSave = TRUE;
}
```

*****/

put_endif ()

```
{
    NumberOfEndifs++;
    indent (CONSEQUENCE);
}
```

```

    strcat (Comp, "endif");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = END_IF;
}
/*****
put_print ()
{
    int i;          /* A simple counter */
    char fault[82]; /* A string to get msg into */

    fault[0] = '\0';
    /*****
    Do the proper indentation if required
    *****/
    indent (CONSEQUENCE);
    strcat (Comp, "print");
    NeedToSave = TRUE;

    /*****
    Get the fault msg class from the user
    *****/
    do
    {
        if(ask("Please enter the fault message class (1-5).", GREEN, fault) == DEFAULT)
            strcpy (fault, "3"); /* The default is 3 dude */
    }
    while (atoi(fault) < 1 || atoi(fault) > 5);

    /*****
    Append the flt msg class to comp
    *****/
    strcat (Comp, fault);
    fault[0] = '\0';
    strcat (Comp, " \042");

    /*****
    Get the fault msg from the user
    *****/
    ask("Please enter fault message. Precede variables by percent sign.\n(e.g. V75T2517A is %V75T2517A)", GREEN, fault);

    /*****
    Append the fault msg to the comp
    *****/
    strcat (Comp, fault); /* Add the print to the Comp string */
    strcat (Comp, "\042"); /* Close the print with a double quote */
    PrevChoice[ChoiceCounter++] = PRINT;
}
/*****
put_comment ()
{
    char comment[MAX_COMMENT_LEN];

    comment[0] = 0;
    ask("Please enter your comment above.", GREEN, comment);
    strcat (Comp, " /* ");
    strcat (Comp, comment);
    strcat (Comp, " */");
    PrevChoice[ChoiceCounter++] = COMMENT;
    NeedToSave = TRUE;
}
/*****
put_string ()
{
    int i, rv;          /* A simple counter */
    char theString1[MAX_STR_LEN], /* A string put the string into */
          theString2[MAX_STR_LEN], /* A string put the string into */
          msg_string[159];

    theString1[0] = '\0';
    theString2[0] = '\0';

    /*****
    Get the string from the user
    *****/
    ask("Please enter your text string above", GREEN, theString1);

    if (Equation == LHS)

```



```

strcpy (CompareType[NumberOfCompares++], "char");

/*****
Clean up indentation
*****/
indent (PREMISE);

/*****
Append the string to the comp
*****/
sprintf (theString2, "\\042%s\\042", theString1);
strcat (Comp, theString2);
NeedToSave = TRUE;
PrevChoice[ChoiceCounter++] = STRING;
}
/*****/
put_eq ()
{
    if (WhereAmI == CONSEQUENCE || NumberOfIfs == NumberOfEndifs)
        strcat (Comp, " =");
    else /* we're in the premise */
        strcat (Comp, " ==");

    PrevChoice[ChoiceCounter++] = EQ;
    NeedToSave = TRUE;
    Equation = RHS;
}
/*****/
put_ge ()
{
    strcat (Comp, " >=");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = GE;
    Equation = RHS;
}
/*****/
put_gt ()
{
    strcat (Comp, " >");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = GT;
    Equation = RHS;
}
/*****/
put_if ()
{
    int i;
    char temp[100];
    /*****/
    If we're balanced then we just add a return
    *****/
    if ((NumberOfIfs - NumberOfEndifs) != 0)
        indent (CONSEQUENCE);
    else strcat (Comp, "\\n");
    strcat (Comp, "if");
    NumberOfIfs++;
    PrevChoice[ChoiceCounter++] = IF;
    WhereAmI = PREMISE;
    Equation = LHS;
    NeedToSave = TRUE;
}
/*****/
put_l_paren ()
{
    char temp[100];

    strcat (Comp, "(");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = L_PAREN;
    /*****/
    If we're defining function arguments we need to increment the function
    paren count so we can tell when we've finished arg def'ing.
    *****/
    if (FuncParenCount > 0)
        FuncParenCount++;
    ParenCount++;
}
/*****/

```

```
put_le ()
{
    strcat (Comp, " <=");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = LE;
    Equation = RHS;
}
/*****/
put_lt ()
{
    strcat (Comp, " <");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = LT;
    Equation = RHS;
}
/*****/
put_multiply ()
{
    strcat (Comp, " *");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = MULTIPLY;
}
/*****/
put_ne ()
{
    strcat (Comp, " <=");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = NE;
    Equation = RHS;
}
/*****/
put_not ()
{
    int i; /* a counter */

    /*****/
    Clean up indentation
    /*****/
    indent (PREMISE);
    strcat (Comp, "not");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = NOT;
    put_l_paren ();
}
/*****/
put_number ()
{
    char number[30], msg_string[159];
    int status, rc;

    /*****/
    Loop till the user gives us a 'good' number
    /*****/
    do
    {
        if (ask("Please enter a number", GREEN, number) == DEFAULT)
            return ERROR;

        /*****/
        str_isalnum returns the type or an error
        /*****/
        if ((status = str_isalnum (number)) == ERROR)
            inform("You must enter a number", RED, 2);

    } while (status == ERROR || rc == ERROR);

    if (Equation == LHS)
    {
        if (status == FLOAT)
            strcpy (CompareType[NumberOfCompares++], "float");
        else strcpy (CompareType[NumberOfCompares++], "int");
    }
    /*****/
    Clean up indentation
    /*****/
}
```

```

    indent (PREMISE);
    strcat (Comp, number);
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = NUMBER;
}
/*****/
put_pi ()
{
    /*****/
    Clean up indentation
    /*****/
    indent (PREMISE);
    strcat (Comp, "PI");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = PI;
}
/*****/
put_func (choice)
int choice;
{
    int i, /* Simple Counter */
        rc; /* A good place to store return codes */
    char whichFunction[FUNC_NAME_LEN]; /* The actual name of the selected function */
    char message[250], reply[50];

    switch(choice)
    {
        case COS: strcpy(FunctionList[FunctionCount], "cos");
            FunctionArguments[FunctionCount] = 1;
            break;

        case ACOS: strcpy(FunctionList[FunctionCount], "acos");
            FunctionArguments[FunctionCount] = 1;
            break;

        case SIN: strcpy(FunctionList[FunctionCount], "sin");
            FunctionArguments[FunctionCount] = 1;
            break;

        case ASIN: strcpy(FunctionList[FunctionCount], "asin");
            FunctionArguments[FunctionCount] = 1;
            break;

        case TAN: strcpy(FunctionList[FunctionCount], "tan");
            FunctionArguments[FunctionCount] = 1;
            break;

        case ATAN: strcpy(FunctionList[FunctionCount], "atan");
            FunctionArguments[FunctionCount] = 1;
            break;

        case SQRT: strcpy(FunctionList[FunctionCount], "sqrt");
            FunctionArguments[FunctionCount] = 1;
            break;

        case POWER: strcpy(FunctionList[FunctionCount], "power");
            FunctionArguments[FunctionCount] = 2;
            break;

        case LOG: strcpy(FunctionList[FunctionCount], "log");
            FunctionArguments[FunctionCount] = 1;
            break;

        case EXP: strcpy(FunctionList[FunctionCount], "exp");
            FunctionArguments[FunctionCount] = 1;
            break;

        case FUNCTION: /*****/
            Get the function name.
            /*****/
            do
            {
                rc = ask("Please enter the function name.\nQ to quit this prompt.", GREEN, whichFunction);
                if(whichFunction[0]=='q' || whichFunction[0]=='Q' || rc==DEFAULT)
                    return DEFAULT;
                if ((rc = funcCheck (whichFunction)) == ERROR)
                {
                    sprintf (message, "'%s' is not a defined user function.\nHit [RETURN] to continue.", whichFunction);
                }
            } while (rc != DEFAULT);
    }
}

```

```
hFunction);
        ask(message, RED, reply);
    }
    while(rc == ERROR);
    strcpy(FunctionList[FunctionCount], whichFunction);
    FunctionArguments[FunctionCount] = MAX_FUNC_PARAMS;
    break;

    default:    break;
}

FunctionCurrent = FunctionCount;
FunctionArgsDef[FunctionCount] = 0;

/*****
Clean up indentation
*****/
indent (PREMISE);
strcat (Comp, FunctionList[FunctionCount++]);
PrevChoice[ChoiceCounter++] = choice;
put_l_paren ();
/*****
We're now in a function so increment the function
paren count if it wasn't already incremented by put_l_paren.
(It will be incremented in put_l_paren if it's a nested
function.)
*****/
if(FuncParenCount == 0)
    FuncParenCount++;
NeedToSave = TRUE;
}
/*****/
put_comma()
{
    strcat(Comp, ",");
    FunctionArgsDef[FunctionCurrent]++;
    PrevChoice[ChoiceCounter++] = COMMA;
    NeedToSave = TRUE;
}
/*****/
put_or ()
{
    strcat (Comp, " or");
    PrevChoice[ChoiceCounter++] = OR;
    Equation = LHS;
    NeedToSave = TRUE;
}
/*****/
put_bitOr ()
{
    strcat (Comp, " bitOr");
    PrevChoice[ChoiceCounter++] = BITOR;
    NeedToSave = TRUE;
}
/*****/
put_bitAnd ()
{
    strcat (Comp, " bitAnd");
    PrevChoice[ChoiceCounter++] = BITAND;
    NeedToSave = TRUE;
}
/*****/
put_shiftL ()
{
    strcat (Comp, " shiftL");
    PrevChoice[ChoiceCounter++] = SHIFTL;
    NeedToSave = TRUE;
}
/*****/
put_shiftR ()
{
    strcat (Comp, " shiftR");
    PrevChoice[ChoiceCounter++] = SHIFTR;
    NeedToSave = TRUE;
}
/*****/
put_bitXor ()
```

```
{
    strcat (Comp, " bitXor");
    PrevChoice[ChoiceCounter++] = BITXOR;
    NeedToSave = TRUE;
}
/*****/

put_r_paren ()
{
    int i;

    strcat (Comp, " )");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = R_PAREN;
    /*****/
    If we're defining function arguments
    we need to dercrement the function
    paren count so we can tell when we've
    finished arg def'ing.
    *****/
    if(FuncParenCount > 0)
    {
        FuncParenCount--;
        if (++FunctionArgsDef[FunctionCurrent] == FunctionArguments[FunctionCurrent])
        {
            for (i=FunctionCurrent-1; i >= 0; i--)
            {
                if (FunctionArgsDef[i] != FunctionArguments[i]);
                {
                    FunctionCurrent = i;
                    break;
                }
            }
        }
        ParenCount--;
    }
}
/*****/

put_set ()
{
    int i; /* A counter */

    /*****/
    Clean up indentation
    *****/
    indent (CONSEQUENCE);
    strcat (Comp, "set");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = SET;
    Equation = LHS;
}
/*****/

put_subtract ()
{
    strcat (Comp, "-");
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = SUBTRACT;
}
/*****/

put_then ()
{
    int i; /* a counter */

    NestedElseCheck[NumberOfIfs-NumberOfEndifs] = THEN;
    /*****/
    Clean up indentation
    *****/
    strcat (Comp, "\n");
    for (i=0; i<((NumberOfIfs-NumberOfEndifs)-1)*SIZE_INDENT; i++)
        strcat (Comp, " ");
    strcat (Comp, "then");
    PrevChoice[ChoiceCounter++] = THEN;
    WhereAmI = CONSEQUENCE;
    Equation = LHS;
    NeedToSave = TRUE;
}
```

90/06/07
16:53:08

remove.c

1

REMOVE

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/27/88

Version: 1.0

Project: CODE (Comp Development Environment)
IESP (INCO Expert System Project)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/

/*****
    Include files
*****/
#include "code.h"
#include "color.h"

remove ()
{
    FILE *ptr1, *ptr2;
    int i,j,k; /* Simple Counters */
    int is_installed; /* TRUE if the comp being removed was previously installed, else FALSE */
    int signal_index; /* The index into the signal table for a variable of type signal */
    int index; /* The index returned from var_check */
    int returnValue,rc; /* return values for function calls */
    char group_dat_file[PATH_LEN]; /* CompInfo file name */
    char file_name[80], confirm[250], cmd_string[250], temp[150];

    rc=ask("Do you want to remove a (G)roup, a (C)omp, or just (Q)uit this?",GREEN,confirm);

    if(rc == DEFAULT || confirm[0] != 'c' && confirm[0] != 'C' &&
       confirm[0] != 'g' && confirm[0] != 'G')
    {
        inform("",BLUE,0);
        return OK;
    }

    /*****
        REMOVE A COMP
    *****/
    if (confirm[0] == 'c' || confirm[0] == 'C')
    {
        rc = get_name (NumOfGroups, "Group", GroupName, &GroupNumber);
        if (rc == ERROR)
            return(ERROR);
        rc = get_name (NumOfComps, "Comp", CompName, &CompNumber);
        if (rc == ERROR)
            return(ERROR);
        retrieve();
        /*****
            Give the user one more chance to abort this drastic action.
        *****/
        clearWA();
        displayWA(Comp);
        sprintf(confirm, "Are you sure you want to remove %s (Y/N)", CompName);
        if(ask(confirm, GREEN, temp)==DEFAULT || !(temp[0] == 'Y' || temp[0] == 'y'))
            return;

        /*****
            Remove the comp from the <group>.dat file.
        *****/
        for(i=0;i<NumOfComps;i++)
        {
            if(strcmp(CompName, CompInfo[i].name) == 0)
```

```
{
    if(CompInfo[i].disposition == INSTALLED)
        is_installed = TRUE;
    else is_installed = FALSE;
    for (j=i;j<NumOfComps;j++)
    {
        CompInfo[j] = CompInfo[j+1];
    }
    break;
}
}
NumOfComps--;

/*****
Save the CompInfo.
*****/
sprintf(group_dat_file, "%s/%s.dat", AMSupport, GroupName);
if (!(ptr1 = fopen (group_dat_file, "w")))
{
    sprintf (cmd_string, "You do not have r/w permission to save %s.dat -- Hit [RETURN]", GroupName);
    ask (cmd_string, GREEN, cmd_string);
    return (ERROR);
}
fprintf (ptr1, "%name_name noise_filter rate on_off disposition\n");
for (i=0;i < NumOfComps;i++)
{
    fprintf (ptr1,"%s %d %d %d %d %s\n", CompInfo[i].name,
        CompInfo[i].noise_filter,
        CompInfo[i].rate,
        CompInfo[i].on_off,
        CompInfo[i].disposition,
        CompInfo[i].purpose);
}
fclose(ptr1);

/*****
Remove the high_level, c, and variable file from the
group directory.
*****/
sprintf (cmd_string,"rm %s/%s/%s.* 2>>/tmp/code.err",CodeGroups,GroupName,CompName);
system (cmd_string);
sprintf(cmd_string, "Comp %s has been removed", CompName);
sleep(2);

/*****
If the comp was previously installed, we need to
install the group again using the new '.dat' file.
*****/
if (is_installed)
{
    install();
}
}

/*****
REMOVE A GROUP
*****/
else if (confirm[0] == 'g' || confirm[0] == 'G')
{
    rc = get_name (NumOfGroups, "Group", GroupName, &GroupNumber);
    if (rc == ERROR)
        return(ERROR);

    /*****
Give the user one more chance to abort this action.
*****/
    sprintf(confirm, "Are you sure you want to remove %s (Y/N)", GroupName);

    do
    {
        if((rc=ask (confirm, GREEN, confirm)) == DEFAULT)
            confirm[0] = 'N';
    }while(!(confirm[0] == 'N' || confirm[0] == 'n' ||
        confirm[0] == 'Y' || confirm[0] == 'y'));

    if(!(confirm[0] == 'Y' || confirm[0] == 'y'))
        return;

    /*****
```

30/06/07
16:53:08

remove.c

3

```
So far so good --- remove that group and it's .dat
and the executable.
*****/
sprintf (cmd_string, "rm -r %s/%s %s/%s.dat %s/%s 2>>/tmp/code.err", CodeGroups, GroupName, AMSupport, Gr
oupName, AMGroups, GroupName);
system (cmd_string);
/*
 * Remove the bugger from the group_names list by sliding everybody
 * else up a notch.
 */
for (i=0; i<NumOfGroups; i++)
{
    if (!strcmp (GroupName, GroupInfo[i].name))
    {
        for (j=i; j<NumOfGroups-1; j++)
        {
            GroupInfo[j].disposition = GroupInfo[j+1].disposition;
            strcpy (GroupInfo[j].name, GroupInfo[j+1].name);
        }
        break;
    }
}
NumOfGroups--; /* Take away one for the one just dropped */

/*
 * Write back the changes to the GroupNames file
 */
writeGroupNames();

sprintf (confirm, "Group %s has been deleted", GroupName);
inform (confirm, PURPLE, 2);
}

/*
 * Since we've just deleted what's in the work area let's do
 * some house cleaning.
 */
ChoiceCounter = 0;
Comp[0] = '\0';
displayWA (Comp);
/*
 * Set up the previous choice so the call to color valid
 * in code.c will only allow the choices we allow on startup.
 */
PrevChoice[ChoiceCounter++] = 99;
mgihue (BLACK);
mgrbox (0.70, 0.9120, 0.999, 0.9320);
}
```


90/06/07
16:55:44

retrieve.c

1

/*****

RETRIEVE

Purpose: Retrieve reads in the comp variable file into the
CompVars struct, reads the comp language file into
a temporary string which is passed to pretty_comp()
for formatting and history set-up.

Returns: 0 - OK, no problems.
-1 - File i/o problem.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/10/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

Include files

*****/

#include "code.h"

```
retrieve ()
{
    FILE      *ptr, *ptr1, *ptr2; /* File pointers, what else could they be */

    char      msg_string[MAX_MSG_LEN], /* Message string repository, and temp string holder */
              tempComp[MAX_COMP_LEN], /* Used for temp storage of the comp file */
              variable_file[PATH_LEN], /* The string for the path of the comps variable file */
              ch, /* For getting a character */
              high_level_file[PATH_LEN], /* The string for the path of CODE language file */
              sig_tbl_type[TYPE_LEN], /* Temporary place for signal table type */
              header[80]; /* Holds the strings that comprise the signal table header. */

    int      i, /* A lovely little counter */
             return_value; /* Return value from fscanf for stripping signal table header */

    /*****
    Initialize some global variables
    *****/
    cleanSlate();
    SignalCount = 0;

    /*
    * Load the signal list and info for signal validation in put_signal.c
    */
    if(ptr = fopen (SignalTbl,"r"))
    {
        /*
        * Rip the header off the file
        */
        do
        {
            return_value = fscanf (ptr, "%s", header);
        } while (!(return_value == EOF || strcmp(header, FIRST_SIGNAL) == 0));
        if(return_value != EOF)
        {
            strcpy(SignalTable[SignalCount].name, header);
            fscanf (ptr, "%s", sig_tbl_type);
            if (sig_tbl_type[0] == 'S')
                strcpy(SignalTable[SignalCount].type, "c");
            else strcpy(SignalTable[SignalCount].type, sig_tbl_type);
            fscanf(ptr, "%[^\\n]", SignalTable[SignalCount].nomenclature);
            SignalCount++;
            /*****
            Loop through the SignalTable file reading the first
            three fields.
            *****/
            while ((fscanf (ptr, "%s", SignalTable[SignalCount].name)) != EOF)
            {
                /*****
                Get the signal table type. If it's a string the
                signal table will have an 'S' - CODE expects a 'char'
                *****/
                fscanf (ptr, "%s", sig_tbl_type);
                if (sig_tbl_type[0] == 'S')
                    strcpy(SignalTable[SignalCount].type, "char");
                else strcpy(SignalTable[SignalCount].type, sig_tbl_type);
                /*****
                Get the nomenclature.
                *****/
                fscanf(ptr, "%[^\\n]", SignalTable[SignalCount].nomenclature);
                ++SignalCount;
            }
        }
        fclose (ptr);
    }

    /*****
    Let's check for the comp var file and read them if they exist.
    *****/
    sprintf (variable_file, "%s/%s/%s.v", CodeGroups, GroupName, CompName);
    if (!(ptr2 = fopen (variable_file, "r")))
    {
        ask("Variables have not been found for this comp -- Hit [Return] to continue.", GREEN, msg_string);
        return(ERROR);
    }
    else
    {
        /*****

```

```

First we will discard the comment
that is the 1st line of all variable files.
*****/
fscanf(ptr2, "%*[^\n]");
NumCompVars = 0;
/*****
Now let's read the variable info until
we reach the end of the file.
*****/
while(fscanf (ptr2,"%s %s %s %d %d %f %f %f %f %[^\n]",
    CompVars[NumCompVars].name,
    CompVars[NumCompVars].type,
    CompVars[NumCompVars].class,
    &CompVars[NumCompVars].occurrence,
    &CompVars[NumCompVars].put_or_get,
    &CompVars[NumCompVars].lo1_limit,
    &CompVars[NumCompVars].lo2_limit,
    &CompVars[NumCompVars].hi1_limit,
    &CompVars[NumCompVars].hi2_limit,
    CompVars[NumCompVars].nomenclature) != EOF)
{
    NumCompVars++;
}
fclose (ptr2);
}

/*
 * Read the comp into the global string Comp
 */
if (readCODEFile(CompName, tempComp) == ERROR)
{
    ask("There was an error reading the comp. Hit [Return] to continue.", GREEN, msg_string);
    return (ERROR);
}

/*
 * Reformat the comp with pretty_comp and
 * set up the choice history
 */
if (pretty_comp (tempComp) == ERROR)
{
    sprintf (msg_string, "Some tokens are undefined in this comp; Corrections must occur before installation
is possible.\nHit [RETURN] to continue.");
    ask(msg_string, GREEN, msg_string);
    CompInfo[CompNumber].disposition = ERROR;
    save ();
    return (ERROR);
}
NeedToSave = FALSE;
return (OK);
} /* End of Retrieve */

```

```
/*  
*****  
PRETTY_COMP
```

Purpose: Pretty_comp takes a temp_comp string from retrieve and formats it nicely. It also sets up the ChoiceCounter integer array which is a token history.

NOTE: This routine treats the comp file like a script and recreates the comp in the screen the same way as if a user were mousing tokens off the screen. The exceptions are: prints, comments, variables, numbers.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/26/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/  
pretty_comp (rawComp)  
  char *rawComp; /* The string to fix and put into global comp */  
{  
  int      rawIndex, /* The index used with the passed string */  
          faultNum, /* The number associated with a print msg */  
          i, /* A simple index */  
          local_index, /* A local index for a variable set by var_check */  
          global_index, /* A global index for a variable set by var_check */  
          tokenIndex; /* The index used with the token string */  
  
  char      token[1000], /* Temporary holding string for a token */  
            temp1[150], /* Always can find a use for a temp string */  
            temp2[150], /* That's right bubba! */  
            msg_string[300],  
            arg[MAX_VAR_LEN+4]; /* Contains the argument to the trig functions */  
  
  /*  
  *****  
  Loop through the CODE language string, translating  
  tokens to C, until the end of the string is reached.  
  *****  
  */  
  NumberOfIfs = rawIndex = 0;  
  NumberOfEndifs = 0;  
  /*  
  *****  
  Read tokens until you run out ...  
  *****  
  */  
  while (rawComp[rawIndex])  
  {  
    /*  
    *****  
    Start with a fresh token index  
    *****  
    */  
    tokenIndex = 0;  
    /*  
    *****  
    Yank out the control characters and spaces,  
    we don't care about them spaces and such.  
    *****  
    */  
    while (rawComp[rawIndex] == ' ' || rawComp[rawIndex] == '\t' ||  
           rawComp[rawIndex] == '\n')  
    {  
      token[tokenIndex++] = rawComp[rawIndex++];  
    }  
    token[tokenIndex] = 0;  
    strcat (Comp, token);  
    tokenIndex = 0;  
  
    /*  
    *****  
    YANK A TOKEN out of the comp's string. Tokens  
    are delimited by spaces, tabs, or line feeds.  
    *****  
    */  
  }  
}
```

```
*****/
while (rawComp[rawIndex] != ' ' && rawComp[rawIndex] != '\t' &&
      rawComp[rawIndex] != '\n' && rawComp[rawIndex] != 0)
{
    token[tokenIndex++] = rawComp[rawIndex++];
}
token[tokenIndex] = 0;

/*****
If the token is null we've reached the end of
the line and need to stop token processing
*****/
if (!token[0]) return (OK);

/*****
      COMMENTS
*****/
if (strcmp(token, "/*", 2) == 0)
{
    /*****
    If this comment is not complete we need to get the rest.
    *****/
    if (token[strlen(token)-1] != '/')
    {
        token[tokenIndex] = rawComp[rawIndex];
        while (!(rawComp[rawIndex-1] == '*' && rawComp[rawIndex] == '/'))
            token[tokenIndex++] = rawComp[rawIndex++];
        token[tokenIndex] = 0;
    }
    /*****
    Now add the comment token to the end of the CODE string.
    *****/
    strcat (Comp, token);
    token[0] = 0;
    PrevChoice[ChoiceCounter++] = COMMENT;
}
/*****
      PRINT
*****/
else if (strcmp(token, "print", 5) == 0)
{
    strcat (Comp, token);
    /*****
    Get all the chars delimited by double quotes
    *****/
    tokenIndex = 0; /* Start again for the message */
    token[tokenIndex] = 0; /* null that string */
    for (i=0; i<2; i++)
    {
        do /* Grab chars until the double quote */
        {
            token[tokenIndex++] = rawComp[rawIndex++];
        } while (rawComp[rawIndex-1] != '"');
        token[tokenIndex] = 0;
        /*****
        Let's add it to the Comp string
        *****/
        strcat (Comp, token);
        token[0] = 0;
        PrevChoice[ChoiceCounter++] = PRINT;
    }
}
/*****
      STRING
*****/
else if (token[0] == '"')
{
    /*****
    Get rest of the string if need be
    *****/
    if (token[strlen(token)-1] != '"')
    {
        do /* Grab chars until the double quote */
        {
            token[tokenIndex++] = rawComp[rawIndex++];
        } while (rawComp[rawIndex-1] != '"');
    }
}
```

```
token[tokenIndex] = 0;
/*****
Let's add it to the comp string
*****/
strcat (Comp, token);
token[0] = 0;
PrevChoice[ChoiceCounter++] = STRING;
)
/*****
IF
*****/
else if (strcmp(token,"if") == 0)
{
    strcat (Comp, token);
    WhereAmI = PREMISE;
    Equation = LHS;
    NumberOfIfs++;
    PrevChoice[ChoiceCounter++] = IF;
}
/*****
ENDIF
*****/
else if (strcmp(token,"endif") == 0)
{
    strcat (Comp, token);
    NumberOfEndifs++;
    PrevChoice[ChoiceCounter++] = END_IF;
}
/*****
THEN
*****/
else if (strcmp(token,"then") == 0)
{
    strcat (Comp, token);
    NestedElseCheck[NumberOfIfs-NumberOfEndifs] = THEN;
    PrevChoice[ChoiceCounter++] = THEN;
    WhereAmI = CONSEQUENCE;
    Equation = LHS;
}
/*****
ELSE
*****/
else if (strcmp(token,"else") == 0)
{
    strcat (Comp, token);
    NestedElseCheck[NumberOfIfs-NumberOfEndifs] = ELSE;
    PrevChoice[ChoiceCounter++] = ELSE;
    Equation = LHS;
}
/*****
OR
*****/
else if (strcmp(token,"or") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = OR;
    Equation = LHS;
}
/*****
BITXOR
*****/
else if (strcmp(token,"bitXor") == 0 || strcmp(token,"xor") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = BITXOR;
}
/*****
BITOR
*****/
else if (strcmp(token,"bitOr") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = BITOR;
}
/*****
BITAND
*****/
else if (strcmp(token,"bitAnd") == 0)
```

```
(
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = BITAND;
)
/*****
    SHIFTL
    *****/
else if (strcmp(token,"shiftL") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SHIFTL;
}
/*****
    SHIFTR
    *****/
else if (strcmp(token,"shiftR") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SHIFTR;
}
/*****
    AND
    *****/
else if (strcmp(token,"and") == 0)
{
    if (WhereAmI == PREMISE)
    {
        strcat (Comp, token);
        PrevChoice[ChoiceCounter++] = AND;
        Equation = LHS;
    }
}
/*****
    NOT
    *****/
else if (strcmp(token,"not") == 0)
{
    PrevChoice[ChoiceCounter++] = NOT;
    strcat (Comp, token);
}
/*****
    SQRT
    *****/
else if (strcmp(token,"sqrt") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SQRT;
}
/*****
    POWER
    *****/
else if (strcmp(token,"power") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = POWER;
}
/*****
    EXP
    *****/
else if (strcmp(token,"exp") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = EXP;
}
/*****
    LOG
    *****/
else if (strcmp(token,"log") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = LOG;
}
/*****
    PI
    *****/
else if ((strcmp(token,"p") == 0) || (strcmp(token, "PI") == 0))
{
    strcat (Comp, token);

```

```
    PrevChoice[ChoiceCounter++] = PI;
}
/*****
    COS
*****/
else if (strcmp(token,"cos") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = COS;
}
/*****
    ACOS
*****/
else if (strcmp(token,"acos") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ACOS;
}
/*****
    SIN
*****/
else if (strcmp(token,"sin") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SIN;
}
/*****
    ASIN
*****/
else if (strcmp(token,"asin") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ASIN;
}
/*****
    TAN
*****/
else if (strcmp(token,"tan") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = TAN;
}
/*****
    ATAN
*****/
else if (strcmp(token,"atan") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ATAN;
}
/*****
    LT
*****/
else if (strcmp(token,"<") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = LT;
    Equation = RHS;
}
/*****
    GT
*****/
else if (strcmp(token,">") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = GT;
    Equation = RHS;
}
/*****
    ADD
*****/
else if (strcmp(token,"+") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ADD;
}
/*****
    SUBTRACT
```



```
*****/  
else if (strcmp(token, "-") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = SUBTRACT;  
}  
/*****  
    MULTIPLY  
*****/  
else if (strcmp(token, "*") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = MULTIPLY;  
}  
/*****  
    DIVIDE  
*****/  
else if (strcmp(token, "/") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = DIVIDE;  
}  
/*****  
    LE  
*****/  
else if (strcmp(token, "<=") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = LE;  
    Equation = RHS;  
}  
/*****  
    GE  
*****/  
else if (strcmp(token, ">=") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = GE;  
    Equation = RHS;  
}  
/*****  
    NE  
*****/  
else if (strcmp(token, "!=") == 0 || strcmp(token, "<>") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = NE;  
    Equation = RHS;  
}  
/*****  
    EQ (in consequence)  
*****/  
else if (strcmp(token, "==") == 0 || strcmp(token, "!=") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = EQ;  
    Equation = RHS;  
}  
/*****  
    L_PAREN  
*****/  
else if (strcmp(token, "(") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = L_PAREN;  
    if(FuncParenCount > 0)  
        FuncParenCount++;  
    ParenCount++;  
}  
/*****  
    R_PAREN  
*****/  
else if (strcmp(token, ")") == 0)  
{  
    strcat (Comp, token);  
    PrevChoice[ChoiceCounter++] = R_PAREN;  
    if(FuncParenCount > 0)  
        FuncParenCount--;
```

```
    ParenCount--;
}
/*****
    COMMA
*****/
else if (strcmp(token, ",") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = COMMA;
}
/*****
    SET
*****/
else if (strcmp(token, "set") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SET;
    Equation = LHS;
}
/*****
    NUMBERS and VARIABLES
*****/
else
{
    /*****
        Determine whether it was msid, signal, local and set
        the ChoiceCounter token history. If it's not defined
        locally we will assume it's a number.
        *****/
    var_check (token, &local_index, &global_index, "null");
    if (local_index >= 0)
    {
        strcat (Comp, token);
        if (strcmp (CompVars[local_index].class, "msid") == 0)
            PrevChoice[ChoiceCounter++] = MSID;
        else if (strcmp (CompVars[local_index].class, "signal") == 0)
            PrevChoice[ChoiceCounter++] = SIGNAL;
        else if (strcmp (CompVars[local_index].class, "local") == 0)
            PrevChoice[ChoiceCounter++] = LOCAL;
        /*****
            Set the type for this comparison if on the LHS
            *****/
        if (Equation == LHS)
            strcpy (CompareType[NumberOfCompares++], CompVars[local_index].type);
    }
    else
    {
        if (funcCheck (token) != ERROR)
        {
            strcat (Comp, token);
            PrevChoice[ChoiceCounter++] = FUNCTION;
        }
        else if (str_isalnum (token) != ERROR)
        {
            strcat (Comp, token);
            PrevChoice[ChoiceCounter++] = NUMBER;
        }
        else
        {
            /*****
                Flag the unknown token to the user, if
                it has not been flagged before.
                *****/
            if (strncmp (token, "...", 3) != 0)
            {
                strcat (Comp, "...");
                strcat (Comp, token);
                strcat (Comp, "...");
            }
            else strcat (Comp, token);
            CompInfo[CompNumber].disposition = ERROR;
        }
    }
}
} /* end of numbers or variables */
} /* end of while loop */
return (OK); /* We done good buck-a-roo */
} /* end of read_tokens () */
```

90/06/07
16:59:11

save.c

1

SAVE

Purpose: Save is for saving the Comp and its necessary files

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/15/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/
#include "code.h"

save ()
{
    int i, j; /* Counters */

    char explain[200], /* Text message repository */
        highLevelFile[PATH_LEN], /* Path for CODE lang. file */
        compVarFile[PATH_LEN], /* Path for comp variable file */
        groupDatFile[PATH_LEN]; /* Path for the [GroupName].dat file */

    FILE *filePtr, /* file pointers */
        *openFile(); /* A handy file opening routine */

    /*
     * Since we're saving, that means the group has been modified
     * So we better change the groups disposition to COMPLETE and
     * remove the old executable if it had previously been INSTALLED.
     */
    if(GroupInfo[GroupNumber].disposition == INSTALLED)
    {
        GroupInfo[GroupNumber].disposition = COMPLETE;
        sprintf(explain, "rm %s/%s 2>> /tmp/code.err", AMGroups, GroupName);
        system(explain);
    }

    /*
     * Let's rebuild the paths in case the comp name has changed since
     * our last save.
     */
    sprintf (highLevelFile, "%s/%s/%s.h", CodeGroups, GroupName, CompName);
    sprintf (compVarFile, "%s/%s/%s.v", CodeGroups, GroupName, CompName);
    sprintf (groupDatFile, "%s/%s.dat", AMSupport, GroupName);

    /*
     * Open CODE language file and write comp.
     */
    if (!(filePtr = openFile (highLevelFile, "w", "save")))
        return (ERROR);

    /******
     Put the CODE language string into it's file
     *****/
    fprintf (filePtr, "%s", Comp);
    fclose (filePtr);

    /******
     Open comp variable file and write variables
     *****/
    if (!(filePtr = openFile (compVarFile, "w", "save")))
        return (ERROR);

    /******
     Write all of the variables for the comp with header
     *****/
}
```

90/06/07
16:59:11

save.c

2

```
fprintf (filePtr, "%variable_name type class occurrences put_or_get lo1_limit lo2_limit hi1_limit hi2_limit nomenclature\n");
for (i=0;i<NumCompVars;i++)
{
    fprintf(filePtr,"%s %s %s %d %d %f %f %f %f %s\n",
        CompVars[i].name,
        CompVars[i].type,
        CompVars[i].class,
        CompVars[i].occurrence,
        CompVars[i].put_or_get,
        CompVars[i].lo1_limit,
        CompVars[i].lo2_limit,
        CompVars[i].hi1_limit,
        CompVars[i].hi2_limit,
        CompVars[i].nomenclature);
}
fclose (filePtr);

/*****
Open the [GroupName].dat file for writing
*****/
if (!(filePtr = openFile (groupDatFile, "w","save")))
    return (ERROR);

/*****
Write all of the group info for the group with header.
*****/
fprintf (filePtr, "%name_name noise_filter rate on_off disposition\n");
for (i=0;i < NumOfComps;i++)
{
    fprintf (filePtr,"%s %d %d %d %d %s\n", CompInfo[i].name,
        CompInfo[i].noise_filter,
        CompInfo[i].rate,
        CompInfo[i].on_off,
        CompInfo[i].disposition,
        CompInfo[i].purpose);
}
fclose (filePtr);

/*
* Put the group names into the GroupInfo file
*/
if (writeGroupNames () == ERROR)
    return (ERROR);

/*
* Tell the user we have saved their comp.
*/
sprintf (explain,"%s has been saved", CompName);
inform(explain,PURPLE,1);
NeedToSave = FALSE;
return (OK);

) /* end of save */
```

90/06/07
16:59:22

strins.c

1

```
static char ScCsId[] = "@(#)strins.c 2.1 8/17/88 08:55:36";
/*-----
 *
 * Function:      strins
 *
 * Entry specification:
 * char *strins(into, from)
 * register char *into, *from;
 *
 * Description:
 * This routine inserts a string into another string, it is similar to the
 * strcat found in the C library (except strcat inserts after).
 *
 * Inputs:
 * into  Pointer-> target string
 * from  Pointer-> string to insert
 *
 * Returns: Nothing
 *
 * External references:      None
 * Resources used:           None
 * Limitations:              None
 * Assumptions:              None
 *
 * Written by: Tom Silva, Bruce G. Jackson & Associates
 *
 * Traceability:
 * Version  Date      Description
 *-----
 * 1.0      09/01/87  initial version
 *
 * Notes:      None
 *-----*/
/*-----
 *-----*/
char *strins(into, from)
register char *into, *from;
{
    register char *to;
    register int to_size, from_size;

    /*-----
     * Find the size of both strings. Include the zero byte for the size of
     * the 'into' string (the zero byte must be moved with the string).
     * Leave the pointers pointing at the zero bytes of the strings.
     */
    for (from_size = 0; *from; from++, from_size++);

    for (to = into, to_size = 1; *to; to++, to_size++);

    /*-----
     * Slide the 'into' string forward by copying it from it's tail to it's
     * head. It needs to be moved forward by the number of bytes in the 'from'
     * string. Then copy the 'from' string into the open slot.
     */
    for (into = to, to += from_size; to_size > 0; to--, into--, to_size--)
        *to = *into;
    for (from--; from_size > 0; to--, from--, from_size--)
        *to = *from;
}
```

90/06/07
16:59:32

token_help.c

1

WRT_TO_DOC

Purpose: Wrt_to_doc writes to the documentation window. It also creates the documentation window and deletes it after it has been written to. Wrt_to_doc uses "More" to display the documentation.

Designer: Robert Z. McFarland/SDC

Programmer: Robert Z. McFarland/SDC

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

Include files

*****/

#include "color.h"

#include "code.h"

#include <termio.h> /* used for setting up the keyboard */

struct termio orig_tty; /* used for setting up the keyboard */

struct termio raw_tty; /* used for setting up the keyboard */

token_help (token, theMODE)

int token, theMODE;

{

int defaultCase = FALSE,
i; /* What would we do without an i */

char cmd[250],
listString[5000]; /* For in-routine formatted display of character data */

inform(MORE,PURPLE,0); /* mouse documentation line */
clearWA(); /* clears and selects WA window */

/*
* Flush the input buffer that may have
* been filled with garbage while input
* was not expected, and go into orig
* tty mode.
*/

ioctl (fileno (stdout), TCFLSH, 0);
ioctl (fileno (stdout), TCSETAW, &orig_tty);

if (theMODE == HELP)

{

switch (token)

{

case 999: strcpy (cmd,"more -n30 ");
break;

case IF: strcpy (cmd,"more -n30 +/\042*IF\042 ");
break;

case THEN: strcpy (cmd,"more -n30 +/\042*THEN\042 ");
break;

case AND: strcpy (cmd,"more -n30 +/\042*AND\042 ");
break;

case OR: strcpy (cmd,"more -n30 +/\042*OR\042 ");
break;

case NOT: strcpy (cmd,"more -n30 +/\042*NOT\042 ");
break;

case PRINT: strcpy (cmd,"more -n30 +/\042*PRINT\042 ");
break;

case SET: strcpy (cmd,"more -n30 +/\042*SET\042 ");

```
        break;

case GT:      strcpy (cmd,"more -n30 +/\042'>' \042 ");
              break;

case GE:      strcpy (cmd,"more -n30 +/\042'>=' \042 ");
              break;

case LT:      strcpy (cmd,"more -n30 +/\042'<' \042 ");
              break;

case LE:      strcpy (cmd,"more -n30 +/\042'<=' \042 ");
              break;

case EQ:      strcpy (cmd,"more -n30 +/\042'=' \042 ");
              break;

case MSID:    strcpy (cmd,"more -n30 +/\042*MSID\042 ");
              break;

case SIGNAL:  strcpy (cmd,"more -n30 +/\042*SIGNAL\042 ");
              break;

case LOCAL:   strcpy (cmd,"more -n30 +/\042*LOCAL\042 ");
              break;

case NUMBER:  strcpy (cmd,"more -n30 +/\042*NUMBER\042 ");
              break;

case INTEGER: strcpy (cmd,"more -n30 +/\042*INTEGER\042 ");
              break;

case FLOAT:   strcpy (cmd,"more -n30 +/\042*FLOAT\042 ");
              break;

case DOUBLE:  strcpy (cmd,"more -n30 +/\042*DOUBLE\042 ");
              break;

case NEW:     strcpy (cmd,"more -n30 +/\042*NEW\042 ");
              break;

case SAVE:    strcpy (cmd,"more -n30 +/\042*SAVE\042 ");
              break;

case RETRIEVE: strcpy (cmd,"more -n30 +/\042*RETRIEVE\042 ");
              break;

case QUIT:    strcpy (cmd,"more -n30 +/\042*QUIT\042 ");
              break;

case DELETE:  strcpy (cmd,"more -n30 +/\042*DELETE\042 ");
              break;

case ELSE:    strcpy (cmd,"more -n30 +/\042*ELSE\042 ");
              break;

case END_IF:  strcpy (cmd,"more -n30 +/\042*ENDIF\042 ");
              break;

case NE:      strcpy (cmd,"more -n30 +/\042'<>' \042 ");
              break;

case L_PAREN: strcpy (cmd,"more -n30 +/\042'(' \042 ");
              break;

case R_PAREN: strcpy (cmd,"more -n30 +/\042')' \042 ");
              break;

case ADD:     strcpy (cmd,"more -n30 +/\042'+' \042 ");
              break;

case SUBTRACT: strcpy (cmd,"more -n30 +/\042'-' \042 ");
              break;

case MULTIPLY: strcpy (cmd,"more -n30 +/\042multiply\042 ");
              break;

case DIVIDE:  strcpy (cmd,"more -n30 +/\042'/' \042 ");
```

90/06/07
16:59:32

token_help.c

3

```
        break;

    case EDIT:        strcpy (cmd,"more -n30 +/\042*EDIT\042 ");
        break;

    case COMMENT:     strcpy (cmd,"more -n30 +/\042*COMMENT\042 ");
        break;

    case HARDCOPY:     strcpy (cmd,"more -n30 +/\042*HARDCOPY\042 ");
        break;

    case INSTALL:     strcpy (cmd,"more -n30 +/\042*INSTALL\042 ");
        break;

    default:          defaultCase = TRUE;
        inform("You have chosen an undocumented token, please notify the developers.",PURPLE,2);
}

/*
 * Now show 'em the words if we have words that is
 */
if (!(defaultCase))
{
    if(access(CodeDocs,F_OK)!=ERROR)
    {
        strcat (cmd, CodeDocs);
        strcat (cmd," 2>>/tmp/code.err");
        system (cmd);
    }
    else inform("Code documentation does not exist on this machine",PURPLE,2);
}
}

else if (theMODE == LIST)
{
    switch (token)
    {
        case MSID:        sprintf (cmd,"more -n30 %s 2>>/tmp/code.err", MSIDTbl);
            system(cmd);
            break;

        case SIGNAL:      sprintf (cmd,"more -n30 %s 2>>/tmp/code.err", SignalTbl);
            system (cmd);
            break;

        case FUNCTION:    strcpy (listString, "Listing of User Functions\n");
            strcat (listString, "_____\n\n");
            for (i=0;i<NumberOfUserFuncs;i++)
            {
                strcat (listString, UserFuncs[i]);
                strcat (listString, "\n");
            }
            displayWA(listString);
            break;

        default:          defaultCase = TRUE;
            displayWA("A listing is not available for this choice.");
    }
}

/*
 * Go back to raw tty modea and clear the work area.
 */
ioctl (fileno (stdout), TCSETAW, &raw_tty);
ask ("Hit [RETURN] to continue", GREEN, cmd);
clearWA();
}
```


90/06/07
16:59:40

translate.c

1

TRANSLATE

Purpose: Translate takes the "Higher Level" language comp and creates a C file which is compatible with the INCO Algorithm Manager group/comp format. This routine figures out the spacing for the C file, so as to remove a dependency from the "Higher Level" comp file.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 7/6/87

Version: 1.0

Project: INCO Expert System Project

Revised by:

Reasons for Revision:

External Interfaces

```
*****/

/*****
    Include files
    *****/
#include "code.h"
#include <ctype.h>
/*****

    Translate
    *****/
translate (compName, CompVars, numCompVars)
    char compName[]; /* The name of the comp to translate */
    struct var_struct CompVars[]; /* Structure containing the comp variable defs */
    int numCompVars; /* The number of comp variable in the structure */
{
    FILE *ptr, /* Just a plain old file pointer */
        *openFile(); /* A handy file opening routine */

    int compIndex, /* Index to the CODE lang. comp string */
        tokenIndex, /* Index to an individual token */
        clangIndex, /* Index to the C lang. comp */
        spaceIndex, /* Index to spacesEtAl */
        equation, /* Similar to global equation in code.h */
        msidIndex, /* MSID index returned from get_msid_index() */
        numberOfIfs, /* The number of if we have parsed */
        numberOfEndifs, /* The number of endifs we have parsed */
        numberOfPrints, /* The number of print we have parsed */
        numberOfSets, /* The number of sets we have parsed */
        lastToken, /* Indicates the previous token type */
        i, j, /* Simple counters */
        MSIDexist, /* TRUE indicates msid exist in this comp */
        fltColor, /* Used for parsing the fault color number out of the msg */
        givenHead, /* TRUE after we have processed the comp header */
        varIndex, /* The index value set in the function getVarIndex */
        relIndex, /* The relative index of an MSID used in getVarIndex */
        varClass, /* The return value from the function getVarIndex */
        charIndex, /* The index of the characters for a variable in a print str */
        whereAmI, /* The local version of the one defined in code.h */
        numToken, /* TRUE if the first token has been found */
        sigShortArraySize, /* The size of the array of structures for noise filter output */
        sigIntArraySize, /* The size of the array of structures for noise filter output */
        sigFloatArraySize, /* The size of the array of structures for noise filter output */
        sigDoubleArraySize, /* The size of the array of structures for noise filter output */
        sigUnsignedArraySize, /* The size of the array of structures for noise filter output */
        sigLongArraySize, /* The size of the array of structures for noise filter output */
        sigOffShortArraySize, /* The size of the array of structures for noise filter output */
        sigOffIntArraySize, /* The size of the array of structures for noise filter output */
        sigOffFloatArraySize, /* The size of the array of structures for noise filter output */
        sigOffDoubleArraySize, /* The size of the array of structures for noise filter output */
        sigOffUnsignedArraySize, /* The size of the array of structures for noise filter output */
}
```

```

sigOffLongArraySize, /* The size of the array of structures for noise filter output */
sigOffStringArraySize, /* The size of the array of structures for noise filter output */
sigStringArraySize; /* The size of the array of structures for noise filter output */

char clangVersion[MAX_COMP_LEN], /* A string to hold the comp's C language version */
token[500], /* A string to hold a token */
compName_c[PATH_LEN], /* Path for C language file */
comp[MAX_COMP_LEN], /* The all important string for holding the comp string */
tmp1[169], /* What would life be without a few transient strings */
tmp2[169], /* Dito my older brother */
msgString[169], /* What would life be without a few transient strings */
varString[MAX_VAR_LEN], /* The string for pulling vars from prints */
spacesEtAl[150], /* A string to hold the ' ', '\t', and '\n' */
lastVarType[TYPE_LEN], /* The type of the last found variable */
strOperator[4], /* The operator used with a string, usually == or != */
compHeader[1000], /* Temporary storage for the comp's header */
endString[250], /* Used to store variable names in formatted prints */
amSprintfing; /* Flag used to know if inside a sprintf statement */

/*
 * Read the comp into the string comp
 */
if (readCODEFile(compName, comp) == ERROR)
{
    sprintf(msgString, "Install: There was a problem reading the CODE file for %s", compName);
    ask(msgString, GREEN, msgString);
    return(ERROR);
}

/*****
 * Initialize some soon-to-be-used variables
 *****/
sigShortArraySize = sigIntArraySize = sigFloatArraySize = sigLongArraySize = 0;
sigDoubleArraySize = sigUnsignedArraySize = sigStringArraySize = 0;
sigOffShortArraySize = sigOffIntArraySize = sigOffFloatArraySize = 0;
sigOffDoubleArraySize = sigOffUnsignedArraySize = sigOffStringArraySize = 0;
numberOfifs = compIndex = clangIndex = givenHead = sigOffLongArraySize = 0;
numToken = numberOfEndifs = numberOfSets = numberOfPrints = spaceIndex = 0;
clangVersion[0] = compHeader[0] = 0;
amSprintfing = FALSE;

/*****
 * Loop through the CODE language string, translating
 * tokens to C, until the end of the string is reached.
 *****/
do
{
    /*****
     * Yank out the control characters and spaces,
     * we don't care about them spaces and such.
     *****/
    yankBlank (comp, &compIndex);

    /*****
     * YANK A TOKEN out of the Comp's string. Tokens
     * are delimited by spaces, tabs, or line feeds.
     *****/
    yankToken (comp, &compIndex, token, &tokenIndex);

    /*****
     * Now the real fun starts. Match the token we
     * just yanked with the allowable set of CODE
     * tokens and translate appropriately.
     *****/
    COMMENTS
    /*****
     * if (strncmp(token, "/*", 2) == 0)
     * {
     *     /*****
     *      * If this comment is not complete we need to get the rest.
     *      *****/
     *     if (token[strlen(token)-1] != '/')
     *     {
     *         token[tokenIndex] = comp[compIndex];
     *         while (!(comp[compIndex-1] == '*' && comp[compIndex] == '/'))
     *             token[tokenIndex++] = comp[compIndex++];
     *         token[tokenIndex++] = comp[compIndex++];
     *         token[tokenIndex] = 0;
     *     }
     * }

```

```
}
if (!(givenHead))
{
    sprintf (compHeader, "%s\n%s()\n", token, compName);
    givenHead = TRUE;
}
/*****
Else add the comment token to the end of the C lang. string.
*****/
else strcat (clangVersion, token);

lastToken = COMMENT;
}
/*****
IF
*****/
else if (strcmp(token,"if") == 0)
{
    numberOfIfs++;
    /*****
Cat a ';' to close off the last statement
*****/
if (lastToken == MSID || lastToken == R_PAREN)
    strcat (clangVersion, ";");
strcat (clangVersion, "\n");
for (i=0;i<(numberOfIfs - numberOfEndifs);i++)
    strcat (clangVersion, "\t");
/*****
Put in a second left paren to surround the
comparitors and ensure proper precedence
*****/
strcat (clangVersion, "if ( (");
lastToken = IF;
whereAmI = PREMISE;
equation = LHS;
} /* end of IF */
/*****
ENDIF
*****/
else if (strcmp(token,"endif") == 0)
{
    /*****
Cat a ';' to close off the last statement
*****/
strcat (clangVersion, ";\n");
for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
    strcat (clangVersion, "\t");
strcat (clangVersion, ")");
lastToken = END_IF;
numberOfEndifs++;
}
/*****
THEN
*****/
else if (strcmp(token,"then") == 0)
{
    /*****
Figure out and set the indentation
*****/
strcat (clangVersion, " )\n");
for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
    strcat (clangVersion, "\t");
strcat (clangVersion, "(");
equation = LHS;
whereAmI = CONSEQUENCE;
lastToken = THEN;
}
/*****
ELSE
*****/
else if (strcmp(token,"else") == 0)
{
    /*****
Figure out and set the indentation
*****/
strcat (clangVersion, ";\n");
for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
    strcat (clangVersion, "\t");
```

```
        strcat (clangVersion, "\n");
        for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
            strcat (clangVersion, "\t");
        strcat (clangVersion, "else\n");
        for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
            strcat (clangVersion, "\t");
        strcat (clangVersion, "{");
        equation = LHS;
        lastToken = ELSE;
    }
/*****
    OR
*****/
else if (strcmp(token,"or") == 0)
{
    strcat (clangVersion, " ) || (");
    equation = LHS;
    lastToken = OR;
}
/*****
    POWER
*****/
else if (strcmp(token,"power") == 0)
{
    strcat (clangVersion, " pow");
    lastToken = POWER;
}
/*****
    EXP
*****/
else if (strcmp(token,"exp") == 0)
{
    strcat (clangVersion, " exp");
    lastToken = EXP;
}
/*****
    BITOR
*****/
else if (strcmp(token,"bitOr") == 0)
{
    strcat (clangVersion, " |");
    lastToken = BITOR;
}
/*****
    BITAND
*****/
else if (strcmp(token,"bitAnd") == 0)
{
    equation = LHS;
    strcat (clangVersion, " &");
    lastToken = BITAND;
}
/*****
    BITXOR
*****/
else if (strcmp(token,"xor") == 0 || strcmp(token, "bitXor") == 0)
{
    equation = LHS;
    strcat (clangVersion, " ^");
    lastToken = BITXOR;
}
/*****
    AND
*****/
else if (strcmp(token,"and") == 0)
{
    equation = LHS;
/*****
    The following is conditional to support
    an older syntax which had 'and's in the
    CONSEQUENCE
*****/
    if (whereAmI == PREMISE)
        strcat (clangVersion, " ) && (");
    lastToken = AND;
}
/*****
    NOT
```

```
*****/  
else if (strcmp(token,"not") == 0)  
{  
    strcat (clangVersion, "!");  
    lastToken = NOT;  
}  
/*****  
    NOT EQUAL TO  
*****/  
*****/  
else if (strcmp(token,"<>") == 0 || strcmp(token,"8") == 0)  
{  
    if (strcmp (lastVarType, "char") != 0)  
        strcat (clangVersion, " !=");  
    else strcpy (strOperator, " !=");  
    equation = RHS;  
    lastToken = NE;  
}  
/*****  
    ADD  
*****/  
*****/  
else if (strcmp(token,"+") == 0)  
{  
    if (strcmp (lastVarType, "char") == 0 || varClass == STRING)  
        strcat (clangVersion, ",");  
    else strcat (clangVersion, " +");  
    lastToken = ADD;  
}  
/*****  
    SUBTRACT  
*****/  
*****/  
else if (strcmp(token,"-") == 0)  
{  
    strcat (clangVersion, " -");  
    lastToken = SUBTRACT;  
}  
/*****  
    MULTIPLY  
*****/  
*****/  
else if (strcmp(token,"*") == 0)  
{  
    strcat (clangVersion, " *");  
    lastToken = MULTIPLY;  
}  
/*****  
    DIVIDE  
*****/  
*****/  
else if (strcmp(token,"/") == 0)  
{  
    strcat (clangVersion, " /");  
    lastToken = DIVIDE;  
}  
/*****  
    P  
*****/  
*****/  
else if ((strcmp(token,"p") == 0) || (strcmp(token,"PI") == 0))  
{  
    strcat (clangVersion, " M_PI");  
    lastToken = PI;  
}  
/*****  
    EQEQ  
*****/  
*****/  
else if (strcmp(token,"==") == 0)  
{  
    if (strcmp (lastVarType, "char") != 0)  
        strcat (clangVersion, " ==");  
    else strcpy (strOperator, " ==");  
    equation = RHS;  
    lastToken = EQ;  
}  
/*****  
    EQ  
*****/  
*****/  
else if (strcmp(token,"=") == 0)  
{  
    if (whereAmI == PREMISE)  
        strcat (clangVersion, " ==");  
    else if (strcmp (lastVarType, "char") != 0)
```

```
        strcat (clangVersion, " =");
        else strcpy (strOperator, " =");
        equation = RHS;
        lastToken = EQ;
    }
    /*****
    GT
    *****/
    else if (strcmp(token, ">") == 0)
    {
        if (strcmp (lastVarType, "char") != 0)
            strcat (clangVersion, " >");
        else strcpy (strOperator, " >");
        equation = RHS;
        lastToken = GT;
    }
    /*****
    GE
    *****/
    else if (strcmp(token, ">=") == 0)
    {
        if (strcmp (lastVarType, "char") != 0)
            strcat (clangVersion, " >=");
        else strcpy (strOperator, " >=");
        equation = RHS;
        lastToken = GE;
    }
    /*****
    LT
    *****/
    else if (strcmp(token, "<") == 0)
    {
        if (strcmp (lastVarType, "char") != 0)
            strcat (clangVersion, " <");
        else strcpy (strOperator, " <");
        equation = RHS;
        lastToken = LT;
    }
    /*****
    LE
    *****/
    else if (strcmp(token, "<=") == 0)
    {
        if (strcmp (lastVarType, "char") != 0)
            strcat (clangVersion, " <=");
        else strcpy (strOperator, " <=");
        equation = RHS;
        lastToken = LE;
    }
    /*****
    SHIFTL
    *****/
    else if (strcmp(token, "shiftL") == 0)
    {
        strcat (clangVersion, " <<");
        lastToken = SHIFTL;
    }
    /*****
    SHIFTR
    *****/
    else if (strcmp(token, "shiftR") == 0)
    {
        strcat (clangVersion, " >>");
        lastToken = SHIFTR;
    }
    /*****
    LEFT PAREN
    *****/
    else if (strcmp(token, "(") == 0)
    {
        strcat (clangVersion, " (");
        lastToken = L_PAREN;
    }
    /*****
    RIGHT PAREN
    *****/
    else if (strcmp(token, ")") == 0)
    {
```

```

    strcat (clangVersion, " ");
    lastToken = R_PAREN;
}
/*****
    SET
*****/
else if (strcmp(token,"set") == 0)
{
    if(!(lastToken == THEN || lastToken == ELSE || lastToken == END_IF) &&
        (numberOfSets > 0 || numberOfPrints > 0))
        strcat (clangVersion, ";");
    lastToken = SET;
    whereAmI = CONSEQUENCE;
    equation = LHS;
    numberOfSets++;
}
/*****
    PRINT
*****/
else if (strncmp(token, "print", 5) == 0)
{
    /*****
        Put a semi-colon after the last line if need be
        *****/
    if(!(lastToken == THEN || lastToken == ELSE || lastToken == END_IF) &&
        (numberOfSets > 0 || numberOfPrints > 0))
        strcat (clangVersion, ";");
    /*****
        Pull the fault class number
        *****/
    tmp1[0]=token[5];
    tmp1[1]=0;
    fltColor = atoi(tmp1);
    compIndex++; /* Dispose of a space */
    tokenIndex = 0; /* Start again for the message */
    endString[0] = 0;
    varString[0] = 0;
    /*****
        Get the first double quote then loop
        until we find the second.
        *****/
    token[tokenIndex++] = comp[compIndex++];
    do
    {
        token[tokenIndex++] = comp[compIndex++];
        /*****
            Break off here to process variables. Yes its true that
            variables should be preceded by a '%' to be recognized here.
            *****/
        if (comp[compIndex-1] == '%')
        {
            /*****
                Get the whole variable
                *****/
            charIndex = 0;
            while ((comp[compIndex] >= 'a' && comp[compIndex] <= 'z') ||
                (comp[compIndex] >= 'A' && comp[compIndex] <= 'Z') ||
                (comp[compIndex] >= '0' && comp[compIndex] <= '9') ||
                comp[compIndex] == '_')
                varString[charIndex++] = comp[compIndex++];
            varString[charIndex] = 0;
            upper (varString); /* Convert to upper case */
            /*****
                Use getVarIndex to find out the class and index
                *****/
            if ((varClass = getVarIndex(varString, CompVars,
                numCompVars, &varIndex, &relIndex)) != ERROR)
            {
                /*****
                    Determine the var. type
                    *****/
                if (CompVars[varIndex].type[0] == 's')
                    token[tokenIndex++] = 'h';
                else if (CompVars[varIndex].type[0] == 'c')
                    token[tokenIndex++] = 's';
                else if (CompVars[varIndex].type[0] == 'f')
                    token[tokenIndex++] = 'f';
                else if (CompVars[varIndex].type[0] == 'i')

```

```

        token[tokenIndex++] = 'd';
    else if (CompVars[varIndex].type[0] == 'd')
    {
        token[tokenIndex++] = 'l';
        token[tokenIndex++] = 'f';
    }
    else /* Don't know this type */
    {
        sprintf (msgString, "Translate: '%s' has unknown type '%s'; aborting translation...
-- Hit [RETURN]", varString, CompVars[varIndex].type);
        ask(msgString, GREEN, msgString);
        return (ERROR);
    }
    /*****
    MSIDs require an array call with index
    *****/
    if (varClass == MSID)
    {
        sprintf(tmp1, " value[%d]", relIndex);
        strcat (endString, tmp1);
        /*****
        Add a comma expecting another variable
        *****/
        strcat (endString, ",");
    }
    /*****
    SIGNALS & locals can just be appended
    *****/
    else if (varClass == SIGNAL || varClass == LOCAL)
    {
        strcat (endString, varString);
        /*****
        Add a comma expecting another variable
        *****/
        strcat (endString, ",");
    }
    else
    {
        sprintf (msgString, "%s is an unknown variable to comp %s.\nHit [RETURN] to continue.
", varString, compName);
        ask(msgString, GREEN, msgString);
        strcat (clangVersion, varString);
    }
} /* end of variables */
} while (comp[compIndex-1] != '\0');
token[tokenIndex] = 0;
/*****
Remove the last comma from the endString
*****/
endString[strlen(endString)-1] = 0;
/*****
Let's do some pretty formatting
*****/
strcat (clangVersion, "\n");
for (i=0; i<(numberOfIfs - numberOfEndifs)+1; i++)
    strcat (clangVersion, "\t");
/*****
sprintf the faultString
*****/
if (endString[0] == 0)
    sprintf(tmp1, "sprintf(faultString, %s);\n", token);
    else /* we got %'s in the print string to add */
        sprintf(tmp1, "sprintf(faultString, %s, %s);\n", token, endString);
strcat (clangVersion, tmp1);
/*****
Let's do some pretty formatting again
*****/
for (i=0; i<(numberOfIfs - numberOfEndifs)+1; i++)
    strcat (clangVersion, "\t");
/*****
Now issue the fault message
*****/
sprintf(tmp1, "fltmsg_issue_NF (faultString, %d, %d, nf[nf_index], faultStruct)", fltColor, numberOfPr
ints);
strcat (clangVersion, tmp1);
/*****
Increment the number of fault messages for this comp

```



```

*****
numberOfPrints++;
/*****
Set the last token to PRINT
*****/
lastToken = PRINT;
}
/*****
VARIABLES
Right away we make a call to getVarIndex to determine wheter
the token is a variable, number of char string. If it is
we proceed, otherwise goodbye.
*****/
else if ((varClass=getVarIndex(token,CompVars,numCompVars,&varIndex,&relIndex)) !=ERROR)
{
    tmp1[0] = 0;tmp2[0] = 0; /* empty tmp1,tmp2 */
    /*
    * If the previous token was a set, then let's indent
    */
    if (lastToken == SET)
    {
        /*
        * If the type is char then we will be
        * sprintf'ing, otherwise not.
        */
        if (CompVars[varIndex].type[0] == 'c' || varClass == STRING)
            amSprintfing = TRUE;
        else amSprintfing = FALSE;

        /*****
        If we are setting a SIGNAL, then increment
        the size of the array of structures for
        signals of this type.
        *****/
        if (varClass == SIGNAL)
        {
            if (CompVars[varIndex].type[0] == 's') /* short */
                sigShortArraySize++;
            else if (CompVars[varIndex].type[0] == 'i') /* int */
                sigIntArraySize++;
            else if (CompVars[varIndex].type[0] == 'f') /* float */
                sigFloatArraySize++;
            else if (CompVars[varIndex].type[0] == 'd') /* double */
                sigDoubleArraySize++;
            else if (CompVars[varIndex].type[0] == 'l') /* long */
                sigLongArraySize++;
            else if (CompVars[varIndex].type[0] == 'u') /* unsigned */
                sigUnsignedArraySize++;
            else if (CompVars[varIndex].type[0] == 'c') /* string */
                sigStringArraySize++;
            else
            {
                sprintf (msgString, "Translate: %s was of unknown type '%s'...Aborting translation",token
, CompVars[varIndex].type);
                ask(msgString, GREEN, msgString);
                return (ERROR);
            }
        }
        strcat (clangVersion, "\n");
        for (i=0; i<(numberOfIfs - numberOfEndifs)+1; i++)
            strcat (clangVersion, "\t");
    }
    else strcat (clangVersion, " ");

    /*
    * STRINGS (embedded variables handled with %)
    */
    if (varClass == STRING && token[strlen(token)-1] != '"')
    {
        do /* Grab chars until the double quote */
        {
            token[tokenIndex++] = comp[compIndex++];
            /*****
            Break off here to process variables. Yes its true that
            variables should be preceded by a '%' to be recognized here.
            *****/
            if (comp[compIndex-1] == '%')
            {

```

```

/*****
Get the whole variable
*****/
charIndex = 0;
while ((comp[compIndex] >='a' && comp[compIndex] <='z') ||
      (comp[compIndex] >='A' && comp[compIndex] <='Z') ||
      (comp[compIndex] >='0' && comp[compIndex] <='9')) {
    comp[compIndex] == '_'
    varString[charIndex++] = comp[compIndex++];
varString[charIndex] = 0;
upper (varString); /* Convert to upper case */
/*****
Use getVarIndex to find out the class and index
*****/
if ((varClass = getVarIndex(varString, CompVars,
                             numCompVars, &varIndex, &relIndex)) != ERROR)
{
    /*****
    Determine the var. type
    *****/
    if (CompVars[varIndex].type[0] == 's')
        token[tokenIndex++] = 'd';
    else if (CompVars[varIndex].type[0] == 'c')
        token[tokenIndex++] = 's';
    else if (CompVars[varIndex].type[0] == 'f')
        token[tokenIndex++] = 'f';
    else if (CompVars[varIndex].type[0] == 'i')
        token[tokenIndex++] = 'd';
    else if (CompVars[varIndex].type[0] == 'd')
    {
        token[tokenIndex++] = 'l';
        token[tokenIndex++] = 'f';
    }
    else /* Don't know this type */
    {
        sprintf (msgString, "Translate: '%s' has unknown type '%s'; aborting translation
... -- Hit [RETURN]", varString, CompVars[varIndex].type);
        ask(msgString, GREEN, msgString);
        return (ERROR);
    }
/*****
MSIDs require an array call with index
*****/
if (varClass == MSID)
{
    /*
    * comma delimits
    */
    sprintf (tmpl, " value[%d],", relIndex);
    strcat (endString, tmpl);
}
/*****
SIGNALS & locals can just be appended
*****/
else if (varClass == SIGNAL || varClass == LOCAL)
{
    strcat (endString, varString);
    /*
    * Comma delimits
    */
    strcat (endString, ",");
}
else
{
    sprintf (msgString, "%s is an unknown variable to comp %s.\nHit [RETURN] to conti
nue.", varString, compName);
    ask(msgString, GREEN, msgString);
    strcat (clangVersion, varString);
}
}
} while (comp[compIndex-1] != '\0'); /* STRINGS, char variables */
token[tokenIndex] = 0;

/*
* Remove the last comma from the endString
*/
endString[strlen(endString)-1] = 0;

```

```
/*
 * sprintf the faultString
 */
if (endString[0] == 0)
    sprintf(tmp1,"%s;\n", token);
else /* we got %'s in the print string to add */
    sprintf(tmp1,"%s,%s;\n", token, endString);
strcat (clangVersion, tmp1);

/*
 * Set the last token to STRING
 */
lastToken = STRING;
amSprintfing = FALSE; /* re-init the bugger */
} /* STRINGS */
else /* STRINGS with embed space and all other variables */
{
    /*
     * Set the lastVarType in order to flag comparitors (e.g. EQEQ)
     */
    if (varClass == NUMBER)
        strcpy (lastVarType, "int");
    else strcpy (lastVarType, CompVars[varIndex].type);

    /*
     * If the variable is not a char string then cat it.
     */
    if(CompVars[varIndex].type[0]!='c' && varClass!=STRING && !amSprintfing)
    {
        if (varClass == MSID)
            sprintf (tmp1," value[%d]", relIndex);
        else sprintf (tmp1, "%s", token);
    }
    else /* It must be of type char or STRING w/embed space(s) */
    {
        if (whereAmI == PREMISE)
        {
            if (equation == LHS)
            {
                if (varClass == MSID)
                    sprintf (tmp1, "strcmp(value[%d],", relIndex);
                else /* SIGNAL or LOCAL */
                    sprintf (tmp1, "strcmp (%s,", token);
            }
            else /* RHS */
            {
                if (varClass == MSID)
                    sprintf(tmp1,"value[%d]]%s 0",relIndex,strOperator);
                else /* SIGNAL, LOCAL */
                    sprintf (tmp1, "%s)%s 0", token, strOperator);
            }
        }
        else /* If CONSEQUENCE it must be a strcpy */
        {
            if (equation == LHS)
            {
                if(varClass == MSID)
                    sprintf (tmp1, "sprintf(value[%d],", relIndex);
                else sprintf (tmp1, "sprintf (%s,", token);
            }
            else /* RHS */
            {
                amSprintfing = FALSE; /* re-init the bugger */
                if(CompVars[varIndex].type[0]=='c' || varClass==STRING)
                    strcpy(tmp1,"\042%s\042");
                else if(CompVars[varIndex].type[0] == 'i' ||
                    CompVars[varIndex].type[0] == 's' ||
                    varClass == NUMBER)
                    strcpy(tmp1,"\042%d\042");
                else if(CompVars[varIndex].type[0] == 'd')
                    strcpy(tmp1,"\042%lf\042");
                else sprintf(tmp1,"\042%%c\042",CompVars[varIndex].type[0]);
                if (varClass == MSID)
                    sprintf(tmp2,"value[%d]",relIndex);
                else
                    sprintf (tmp2, "%s)", token);
            }
            strcat(tmp1,tmp2);
        }
    }
}
```

```

    }
    strcat (clangVersion, tmp1);
    tmp1[0] = 0; tmp2[0] = 0; /* empty tmp1, tmp2 */
    lastToken = MSID; /* MSID ~ SIGNAL ~ LOCAL ~ NUMBER in this case */
}
/* end of variable */
/*****
UNRECOGNIZED TOKENS
If we have not recognized the token at this point we just drop
it in as is without any processing.
*****/
else
{
    strcat (clangVersion, " /* Unkown token */ ");
    strcat (clangVersion, token);
}
numToken++;
}while (comp[compIndex] != 0); /* Until we reach the end of the string */

if (lastToken != END_IF)
    strcat (clangVersion, ";");
/*****
Now that we have completed our token level translation
let's take care of putting the signals back into shared
memory where its necessary.
*****/
for (i=0; i<numCompVars; i++)
{
    if ((strcmp(CompVars[i].class, "signal")==0) &&
        (CompVars[i].put_or_get == PUT || CompVars[i].put_or_get == PET))
    {
        /*****
        Let's take care of indentation now!
        *****/
        strcat (clangVersion, "\n\t");

        sprintf (tmp2, "\\042%s\\042, %s, %d, nf[nf_index]");
        switch (CompVars[i].type[0])
        {
            case 'c': sprintf(tmp1, "putsig_str_NF (\\042%s\\042,%s,%d,nf[nf_index],sigString);",
                               CompVars[i].name, CompVars[i].name, sigStringArraySize++);
                       break;

            case 's': sprintf(tmp1, "putsig_s_NF (\\042%s\\042,%s,%d,nf[nf_index],sigShort);",
                               CompVars[i].name, CompVars[i].name, sigShortArraySize++);
                       break;

            case 'i': sprintf(tmp1, "putsig_i_NF (\\042%s\\042,%s,%d,nf[nf_index],sigInt);",
                               CompVars[i].name, CompVars[i].name, sigIntArraySize++);
                       break;

            case 'f': sprintf(tmp1, "putsig_f_NF (\\042%s\\042,%s,%d,nf[nf_index],sigFloat);",
                               CompVars[i].name, CompVars[i].name, sigFloatArraySize++);
                       break;

            case 'd': sprintf(tmp1, "putsig_d_NF (\\042%s\\042,%s,%d,nf[nf_index],sigDouble);",
                               CompVars[i].name, CompVars[i].name, sigDoubleArraySize++);
                       break;

            case 'l': sprintf(tmp1, "putsig_l_NF (\\042%s\\042,%s,%d,nf[nf_index],sigLong);",
                               CompVars[i].name, CompVars[i].name, sigLongArraySize++);
                       break;

            case 'u': sprintf(tmp1, "putsig_u_NF (\\042%s\\042,%s,%d,nf[nf_index],sigUnsigned);",
                               CompVars[i].name, CompVars[i].name, sigUnsignedArraySize++);
                       break;

            default : break;
        }
        strcat (clangVersion, tmp1);
    }
}
/*****
Let's clean up by adding a few correctly indented ')'
*****/
if ((numberOfIfs - numberOfEndifs) > 0)
{
    strcat (clangVersion, "\n");
}

```

```
for (i=0;i<(numberOfIfs - numberOfEndifs);i++)
    strcat (clangVersion, "\t");
strcat (clangVersion, "\n");
)
strcat (clangVersion, "\n\n"); /* end of the comp! */

/*****
Now that we know what how many of each of the types of
SIGNALS or faultMsg we can declare their data structures.
*****/
strcat (compHeader, "\n\tchar faultString[120];");
if (numberOfPrints)
{
    sprintf (tmpl, "\n\tstatic struct faultMsgStruct faultStruct[%d];", numberOfPrints);
    strcat (compHeader, tmpl);
}
if (sigStringArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigStringStruct sigString[%d];", sigStringArraySize);
    strcat (compHeader, tmpl);
}
if (sigShortArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigShortStruct sigShort[%d];", sigShortArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffShortArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffShortStruct sigOffShort[%d];", sigOffShortArraySize);
    strcat (compHeader, tmpl);
}
if (sigIntArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigIntStruct sigInt[%d];", sigIntArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffIntArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffIntStruct sigOffInt[%d];", sigOffIntArraySize);
    strcat (compHeader, tmpl);
}
if (sigFloatArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigFloatStruct sigFloat[%d];", sigFloatArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffFloatArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffFloatStruct sigOffFloat[%d];", sigOffFloatArraySize);
    strcat (compHeader, tmpl);
}
if (sigDoubleArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigDoubleStruct sigDouble[%d];", sigDoubleArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffDoubleArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffDoubleStruct sigOffDouble[%d];", sigOffDoubleArraySize);
    strcat (compHeader, tmpl);
}
if (sigLongArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigLongStruct sigLong[%d];", sigLongArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffLongArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffLongStruct sigOffLong[%d];", sigOffLongArraySize);
    strcat (compHeader, tmpl);
}
if (sigUnsignedArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigUnsignedStruct sigUnsigned[%d];", sigUnsignedArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffUnsignedArraySize)
```

```

    sprintf(tmp1, "\n\tstatic struct sigOffUnsignedStruct sigOffUnsigned[%d];", sigOffUnsignedArraySize);
    strcat (compHeader, tmp1);
}

/*****
Loop through the entire variable list for the comp
*****/
for (i=0; i<numCompVars; i++)
{
    /*****
    If we find a local put its type declaration into
    the c string with appropriate indentation.
    *****/
    if (strcmp (CompVars[i].class, "local") == 0)
    {
        /*****
        Char strings are handled a little differently
        *****/
        if (CompVars[i].type[0] == 'c')
        {
            sprintf(tmp1, "\n\tstatic char %s[120];", CompVars[i].name);
        }
        else sprintf (tmp1, "\n\tstatic %s %s;", CompVars[i].type, CompVars[i].name);
        strcat (compHeader, tmp1);
    }
}

/*****
Add a carriage return to lift and separate (pronounce with a long 'a')
*****/
strcat (compHeader, "\n");

/*****
Let up now retrieve the signals necessary to run this comp
*****/
for (i=0; i<numCompVars; i++)
{
    /*****
    If we find a signal look at its type and cat the
    appropriate get_sig call into the header definition
    *****/
    if (strcmp (CompVars[i].class, "signal") == 0)
    {
        if (CompVars[i].put_or_get == GET || CompVars[i].put_or_get == PET)
        {
            if (CompVars[i].type[0] == 'c')
                sprintf (tmp1, "\n\tgetsig (\042%s\042, %s);", CompVars[i].name, CompVars[i].name);
            else sprintf (tmp1, "\n\tgetsig (\042%s\042, %s);", CompVars[i].name, CompVars[i].name);

            strcat (compHeader, tmp1);
        }
    }
}

/*****
Add a carriage return to lift and separate (pronounce with a long 'a')
*****/
strcat (compHeader, "\n");

/*****
FILE I/O
Now that we have completed the clangVersion we need to
write it to its file so it can be used by install.
*****/
sprintf (compName_c, "%s/%s.c", CodeGroups, GroupName, compName);
if (!(ptr = fopen (compName_c, "w", "translate")))
{
    sprintf(msgString, "Translate: CODE could not open %s for writing, aborting translation...-- Hit [RETURN]
", compName_c);
    ask(msgString, GREEN, msgString);
    return (ERROR);
}

/*****
Write the clangVersion into its C file for install to use.
*****/
fprintf (ptr, "%s", compHeader, clangVersion);
fclose (ptr);

```

```
    } /* end of translate() */

    /*****
    yankBlank & yankToken functions
    *****/
    yankBlank (comp, compIndex)
        char *comp;
        int *compIndex;
    {
        /*****
        Yank non-chars until you hit a char
        *****/
        while ((comp[*compIndex] == ' ' || comp[*compIndex] == '\t' ||
                comp[*compIndex] == '\n') && comp[*compIndex] != 0)
        {
            (*compIndex)++;
        }
    }

    yankToken (comp, compIndex, token, tokenIndex)
        char *comp;
        int *compIndex;
        char *token;
        int *tokenIndex;
    {
        /*****
        Init the token to null
        *****/
        *tokenIndex = 0;
        /*****
        Yank char until you hit a non char
        *****/
        while (comp[*compIndex] != ' ' && comp[*compIndex] != '\t' &&
                comp[*compIndex] != '\n' && comp[*compIndex] != 0)
        {
            token[( *tokenIndex )++] = comp[( *compIndex )++];
        }
        token[*tokenIndex] = 0;
    }
```

90/06/07
17:03:44

type_check.c

1

```
#include "code.h"
```

```
type_check(type)
char type;
```

```
{
    int illegal = FALSE;
    char response[5];

    switch(CompareType[NumberOfCompares-1][0])
    {
        case 's':    if(WhereAmI == CONSEQUENCE && type != 's' ||
                        WhereAmI == PREMISE && type == 'c')
                        illegal = TRUE;
                        break;

        case 'f':    if((WhereAmI == CONSEQUENCE &&
                        (type == 'c' || type == 'd' || type == 'h')) ||
                        (WhereAmI == PREMISE && type == 'c'))
                        illegal = TRUE;
                        break;

        case 'i':    if((WhereAmI == CONSEQUENCE &&
                        (type == 'c' || type == 'd' || type == 'f' || type == 'h')) ||
                        (WhereAmI == PREMISE && type == 'c'))
                        illegal = TRUE;
                        break;

        case 'c':    if(WhereAmI == PREMISE && type != 'c')
                        illegal = TRUE;
                        break;

        case 'd':    if(type == 'c')
                        illegal = TRUE;
                        break;

        case 'h':    /* Need to know how translate defines hex types */
                        break;

        default:     break;
    }

    if (illegal)
    {
        return(ERROR);
    }

    if (CompareType[NumberOfCompares-1][0] != type)
    {
        ask ("Warning: Operand types are inconsistent in this expression.\nHit [RETURN] to continue", GREEN, response);
    }
    return(OK);
}
```


90/06/07
17:03:54

utilities.c

1

```

/*****
utilities.c
*****/

/*****
Include files
*****/
#include "code.h"
#include "cursor.h"
#include <ctype.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termio.h>      /* used for setting up the keyboard */

```

```
/******  
    openFile
```

Purpose: openFile opens up a file. If it fails it prints a message to that effect and returns a NULL (0).

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 1/15/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/  
FILE *openFile (fileWPath, readWrite, callingRoutine)  
    char fileWPath[], /* The name of the file, path included */  
        readWrite[], /* The open option, read, write, or append */  
        callingRoutine[]; /* The routine requesting the open file */  
{  
    FILE *filePtr; /* The pointer to the file that it to be opened */  
  
    char explain[250], /* Place to put a message in case of problem */  
        tmpString[32]; /* Used in conjunction with explain */  
  
    /*  
     * Open the file with the option.  
     */  
    if (!(filePtr = fopen (fileWPath, readWrite)))  
    {  
        if (readWrite[0] == 'r')  
            strcpy (tmpString, "reading");  
        else if (readWrite[0] == 'w')  
            strcpy (tmpString, "writing");  
        else if (readWrite[0] == 'a')  
            strcpy (tmpString, "appending");  
        else strcpy (tmpString, "something completely different");  
        sprintf(explain, "%s: Can not open '%s' for %s", callingRoutine, fileWPath, tmpString);  
        inform(explain, PURPLE, 1);  
        return (NULL);  
    }  
    /*  
     * On successful open return a pointer.  
     */  
    return (filePtr);  
}
```

```
/******  
UPPER
```

Purpose: Upper takes a string and converts it into upper case.

Designer: David Moyer/RSOC

Programmer: David Moyer/RSOC

Date: 12/11/86 (CODE Release date)

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by: Harold Taylor/SDC

Reasons for Revision: I edited Dave's program so that it only contained the one utility I needed.

External Interfaces

None.

```
*****/  
upper(string)
```

```
/******  
Global Variables  
*****  
string the string to change to upper case  
*****/
```

```
char *string;
```

```
{  
    int i, n, toupper();  
    char c, *ptr;  
  
    ptr = string;  
    n = strlen(string);  
  
    for ( i=0; i<n; i++, ptr++)  
    {  
        c = *ptr;  
        *ptr = toupper(c);  
    }  
}
```

/*****

HARDCOPY

Purpose: Hardcopy sends the following files to the printing device: comp_file, variable file, the "C" version.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/7/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by: Terri Murphy

Reasons for Revision: IGP vs Sony print routines.

*****/

hardcopy ()

{

char hard_copy[400],
CODELanguage[PATH_LEN], /* Path of high level language file */
compVariables[PATH_LEN], /* Path of comp variable file */
compCLanguage[PATH_LEN], /* Path of comp C language file */
groupCLanguage[PATH_LEN]; /* Path of group C language file */

/*****

Prepare the paths

*****/

sprintf (CODELanguage, "%s/%s/%s.h", CodeGroups, GroupName, CompName);
sprintf (compVariables, "%s/%s/%s.v", CodeGroups, GroupName, CompName);
sprintf (compCLanguage, "%s/%s/%s.c", CodeGroups, GroupName, CompName);
sprintf (groupCLanguage, "%s/%s/%s.c", CodeGroups, GroupName, GroupName);

/*****

Prepare the string

*****/

sprintf (hard_copy, "print %s %s %s %s 2>>/tmp/code.err", CODELanguage, compVariables,
compCLanguage, groupCLanguage);

/*****

Ask UNIX for a little help

*****/

inform("Printing...", PURPLE, 0);
system (hard_copy);
inform("", BLUE, 0); /* Clear it */

)

/*****

USRFUNCS

Purpose: A dummy function to satisfy clips.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/7/88

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

*****/

usrfuncs()

{

}

```
/******  
STR_ISALNUM
```

Purpose: Determines whether a given string is alphanumeric.
A single decimal point is allowed.

Returns: 0 - It was alphanumeric.
1 - It was not!

Designer: Troy Heindel/NASA/MOD

Programmer: Troy Heindel/NASA/MOD

Date: 11/14/88

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****
```

```
str_isalnum (theString)
```

```
char theString; /* The string to look at */
```

```
{  
    int i, /* Can't live without an i */  
    decimalFlag = FALSE; /* TRUE if we've seen a decimal point */
```

```
/******  
Leading zero's are a no-no as they conote octal
```

```
*****/  
if (theString[0] == '0' && strlen(theString) > 1)
```

```
{  
    inform("Leading zeros are not allowed",RED,2);  
    return (ERROR);  
}
```

```
/******  
Loop through the whole bloody string, checking to  
see that each char is an alphanumeric
```

```
*****/  
for (i=0; i<strlen(theString); i++)
```

```
{  
    /******  
    Check to see if the char is an alphanumeric  
    *****/  
    if (!((theString[i] >= 'A' && theString[i] <= 'F') ||  
        (theString[i] >= '0' && theString[i] <= '9')))
```

```
{  
    /******  
    No. Then check to see if it is a decmial point.  
    Only let them use one decmial point per number  
    *****/  
    if (theString[i] == '.' && decimalFlag == FALSE)  
        decimalFlag = TRUE;  
    else return (ERROR);  
}
```

```
}  
if (decimalFlag)  
    return (FLOAT); /* All's well that ends well */  
return (INTEGER); /* All's well that ends well */  
}
```

/*****

INDENT

Purpose: Carries out indentation.

Designer: Troy Heindel/NASA/MOD

Programmer: Troy Heindel/NASA/MOD

Date: 11/27/88

Version: 2.0

Project: CODE (Comp Development Environment)

*****/

indent (theArea)

int theArea; /* Either PREMISE or CONSEQUENCE */

{

int i; /* Can't live without an i */

char temp[111];

/*****

Put the correct number of spaces for the area we are in

*****/

if (theArea == PREMISE)

{

/*****

If we are following some logic we need a carriage
return and spaces, otherwise just a single space

*****/

if (PrevChoice[ChoiceCounter-1] == AND ||

PrevChoice[ChoiceCounter-1] == OR ||

PrevChoice[ChoiceCounter-1] == BITOR ||

PrevChoice[ChoiceCounter-1] == BITAND ||

PrevChoice[ChoiceCounter-1] == BITXOR)

{

strcat (Comp, "\n");

for (i=0;i<((NumberOfIfs-NumberOfEndifs)-1)*SIZE_INDENT+3;i++)

strcat (Comp, " ");

}

else strcat (Comp, " ");

}

else /* CONSEQUENCE */

{

/*****/

If we are following some a THEN or an ELSE then
we need a return and spaces, otherwise just a

single space

*****/

if (PrevChoice[ChoiceCounter-1] != THEN &&

PrevChoice[ChoiceCounter-1] != ELSE)

{

strcat (Comp, "\n");

for (i=0;i<(NumberOfIfs-NumberOfEndifs)*SIZE_INDENT;i++)

strcat (Comp, " ");

}

else strcat (Comp, " ");

}

}

/*****

getVarIndex

Returns:

0 - The given name is of class local or signal.

-1 - The given name was not found in either the
local or group variable lists.

index# -

If name was found as an msid in the group list
then an index is returned.

*****/

getVarIndex(name, CompVars, numCompVars, varIndex, relIndex)

char *name;

struct var_struct CompVars[]; /* Structure containing the comp variable defs */

int numCompVars; /* The number of comp variable in the structure */

int *varIndex; /* The index of the variable; MSID, SIGNAL, LOCAL */

int *relIndex; /* The relative index of the MSID */

{

int i, msid_cnt;

char temp[155];

*relIndex = 0;

*varIndex = 0;

/*****

If the first char is a double quote we know
right away that this is a literal string.

*****/

if (name[0] == '"')

{

return (STRING);

}

/*****

See if the variable is in the variable passed
to translate() from install.

*****/

for(i=0;i<numCompVars;i++)

{

/*****

Try to match the name, then the class

*****/

if (strcmp (CompVars[i].name, name) == 0)

{

if (strcmp (CompVars[i].class, "signal") == 0)

{

*varIndex = i;

return(SIGNAL);

}

else if (strcmp (CompVars[i].class, "local") == 0)

{

*varIndex = i;

return(LOCAL);

}

else if (strcmp (CompVars[i].class, "msid") == 0)

{

*varIndex = i;

break;

}

}

}

/*****

If we did not find it as a SIGNAL or LOCAL then see
if it is an MSID in the CompInfo structure

*****/

msid_cnt = 0;

for (i=0;i<NumGroupVars;i++)

{

/*****

We need to keep track of a relative index for
the msids.

*****/

if (strcmp (GroupVars[i].class, "msid") == 0)

{

```

/*****
Try to match the name, return if so
*****/
if (strcmp (GroupVars[i].name, name) == 0)
{
    *relIndex = msid_cnt; /* the relative msid index */
    return (MSID);
}
msid_cnt++;
}

/*****
Now we know that it was not an MSID, signal, or
local, but now see if it was a number, or a string
*****/
if (str_isalnum (name) != ERROR)
{
    return (NUMBER);
}
return (ERROR);
}

```



```

/*****
changeFont

Puprose: To change the font size.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 12/13/88

Version: 2.0

Project: Real-Time Data System
*****/
changeFont (choice, theMode)
    int choice, /* The area we want to change the font in */
    theMode; /* The size of the font */
{
    if (choice != WORK_AREA)
    {
        /*****
        Delete the old graphics fonts
        *****/
        mgidelf (Font);
        mgidelf (BoldFont);
        mgidelf (ItalicFont);

        /*****
        Change to the chosen graphics font
        *****/
        if (theMode == LARGE)
        {
            mgifetchgf (Font, "9x11");
            mgifetchgf (BoldFont, "9x11_b");
            mgifetchgf (ItalicFont, "9x11_i");
        }
        else if (theMode == MEDIUM)
        {
            mgifetchgf (Font, "7x9");
            mgifetchgf (BoldFont, "7x9_bold");
            mgifetchgf (ItalicFont, "7x9_italic");
        }
        else if (theMode == SMALL)
        {
            mgifetchgf (Font, "5x7");
            mgifetchgf (BoldFont, "5x7_bold");
            mgifetchgf (ItalicFont, "5x7_italic");
        }

        /*****
        Redraw the screen to see the results
        *****/
        background ();
        color_valid (PrevChoice[ChoiceCounter - 1], -2);
    }
    else /* Change the text fonts in the text windows */
    {
        if (theMode == SMALL)
        {
            mgitf (3, SmallTextFont, -1, 1);
            mgitf (4, SmallTextFont, -1, 1);
            mgitf (5, SmallTextFont, -1, 1);
            mgitf (7, SmallTextFont, -1, 1);
        }
        else if (theMode == MEDIUM)
        {
            mgitf (3, MediumTextFont, -1, 1);
            mgitf (4, MediumTextFont, -1, 1);
            mgitf (5, MediumTextFont, -1, 1);
            mgitf (7, MediumTextFont, -1, 1);
        }
        else /* assume (theMode == LARGE) */
        {
            mgitf (3, LargeTextFont, -1, 1);
            mgitf (4, LargeTextFont, -1, 1);
            mgitf (5, LargeTextFont, -1, 1);
        }
    }
}

```

90/06/07
17:03:54

utilities.c

10

```
    mgitf (7, LargeTextFont, -1, 1);  
}  
displayWA(Comp);
```

```
/******  
    writeGroupNames
```

Purpose: writeGroupNames writes the GroupInfo file.

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 1/15/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/  
writeGroupNames()  
{  
    FILE *filePtr; /* You guessed it! */  
  
    int i;  
  
    /******  
    Open the GroupInfo file with the option  
    *****/  
    if (!(filePtr = openFile (GroupNamesFile, "w", "writeGroupNames"))  
        return (ERROR);  
  
    /******  
    On successful open, write the file  
    *****/  
    fprintf (filePtr, "#GroupNames Disposition\n");  
  
    for (i=0;i<NumOfGroups;i++)  
        fprintf (filePtr, "%s %d\n", GroupInfo[i].name, GroupInfo[i].disposition);  
    fclose (filePtr);  
  
    return (OK);
```

```
/******  
    readCODEFile
```

Purpose: readCODEFile reads in the CODE high level file into the string which is passed to it.

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 1/24/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/
```

```
readCODEFile(compName, compStr)
```

```
    char compName[]; /* The name of the comp to get */
```

```
    char *compStr; /* The string to put the comp into */
```

```
{
```

```
    FILE *filePtr; /* You guessed it! */
```

```
    int i;
```

```
    char theCODEFile[PATH_LEN]; /* The path to the CODE file */
```

```
    /******
```

```
    Create the path to open the high level file
```

```
    *****/
```

```
    sprintf (theCODEFile, "%s/%s/%s.h", CodeGroups, GroupName, compName);
```

```
    /******
```

```
    Open the GroupInfo file with the option
```

```
    *****/
```

```
    if (!(filePtr = fopen (theCODEFile, "r", "readCODEFile")))
```

```
        return (ERROR);
```

```
    /******
```

```
    Read the entire comp into the comp string.
```

```
    *****/
```

```
    i=0;
```

```
    while (((compStr[i++] = fgetc (filePtr)) != EOF) && i<MAX_COMP_LEN);
```

```
    fclose (filePtr);
```

```
    compStr[i-1] = '\0';
```

```
    /******
```

```
    Make sure we're not too big.
```

```
    *****/
```

```
    if(i==MAX_COMP_LEN)
```

```
    {
```

```
        ask("readCODEFile: This comp exceeds that maximum comp length.\nPlease notify developers. Hit [RETURN]",  
RED,theCODEFile);
```

```
        return (ERROR);
```

```
    }
```

```
    return (OK);
```

```
}
```

```
/******  
readGroupNames
```

Purpose:

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 2/6/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/  
readGroupNames()  
{
```

```
FILE *filePtr; /* You guessed it! */
```

```
char message[120];
```

```
struct group_info_struct tempGroupInfo;
```

```
int i,j; /* counters */
```

```
NumOfGroups = 0;
```

```
/*
```

```
 * Open the GroupNames file for reading
```

```
*/
```

```
if (filePtr = openFile (GroupNamesFile, "r", "readGroupNames"))
```

```
{
```

```
    /*
```

```
     * Strip the header.
```

```
    */
```

```
    fscanf(filePtr, "%*[^\\n]");
```

```
    while (fscanf (filePtr, "%s %d", GroupInfo[NumOfGroups].name,  
                  &GroupInfo[NumOfGroups].disposition) != EOF)
```

```
        ++NumOfGroups;
```

```
    fclose (filePtr);
```

```
    /*
```

```
     * Sort the little buggers so they're nice to look at.
```

```
    */
```

```
    for (i=0; i<NumOfGroups-1; i++)
```

```
    {
```

```
        for (j=i+1; j<NumOfGroups; j++)
```

```
        {
```

```
            if (strcmp (GroupInfo[i].name, GroupInfo[j].name) > 0)
```

```
            {
```

```
                tempGroupInfo = GroupInfo[i];
```

```
                GroupInfo[i] = GroupInfo[j];
```

```
                GroupInfo[j] = tempGroupInfo;
```

```
            }
```

```
        }
```

```
    }
```

```
    return (NumOfGroups);
```

```
}
```

```
/*
```

```
 * Maybe this is a new system and the file does not exist yet.
```

```
 * Try creating the file by opening it for writing.
```

```
*/
```

```
else if (!(filePtr = openFile (GroupNamesFile, "w", "readGroupNames")))
```

```
{
```

```
    sprintf (message, "Unable to access the %s file in order to obtain\na listing of the groups. Hit [RE  
TURN] to continue", GroupNamesFile);
```

```
    ask(message, GREEN, message);
```

```
    return (ERROR);
```

```
}
```

```
else
```

```
{
```

```
    inform("No algorithms were found on this machine.", PURPLE, 2);
```

```
    fclose (filePtr);
```

```
    return (0); /* The number of Groups on this system */
```

```
}
```

```
}
```

```

/*****
readCompNames
*****/

```

Purpose:

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 2/6/89

Version: 2.0

Project: CODE (Comp Development Environment)

```

*****
readCompNames()
{
    FILE *filePtr; /* You guessed it! */

    char file_str[PATH_LEN];

    int i,j;

    struct comp_info_struct tempCompInfo;

    NumOfComps = 0;
    /*
     * Let's open the [GroupName].dat file to find out
     * which comps exist for this group.
     */
    sprintf (file_str, "%s/%s.dat", AMSupport, GroupName);
    if (!(filePtr = openFile (file_str, "r", "readCompNames")))
        return (NumOfComps);

    /*
     * ... and read the CompName, noise_filter, rate,
     * on_off and disposition into an array of group
     * information structures.
     */
    else
    (
        /*
         * Throw out the header comment.
         */
        fscanf(filePtr, "%*[\n]");
        /*
         * Read in what's left.
         */
        while ((fscanf (filePtr, "%s %d %d %d %d %s[\n]", CompInfo[NumOfComps].name,
                        &CompInfo[NumOfComps].noise_filter,
                        &CompInfo[NumOfComps].rate,
                        &CompInfo[NumOfComps].on_off,
                        &CompInfo[NumOfComps].disposition,
                        CompInfo[NumOfComps].purpose)) != EOF)
        {
            NumOfComps++;
        }
        fclose (filePtr);
        /*****
         * Sort the little buggers so they're nice to look at.
         *****/
        for (i=0;i<NumOfComps-1;i++)
        {
            for (j=i+1;j<NumOfComps;j++)
            {
                if (strcmp (CompInfo[i].name, CompInfo[j].name) > 0)
                {
                    tempCompInfo = CompInfo[i];
                    CompInfo[i] = CompInfo[j];
                    CompInfo[j] = tempCompInfo;
                }
            }
        }
    )
    return (NumOfComps);
}

```

```
/******  
cleanExit
```

Purpose: cleanExit is used to clean up ttymodes and graphics
before exiting CODE.

Designer: Terri Murphy/NASA/JSC/MOD

Programmer: Terri Murphy/NASA/JSC/MOD

Date: 1/24/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/  
cleanExit()  
{  
  
    struct termio orig_tty;    /* used for setting up the keyboard */  
    struct termio raw_tty;     /* used for setting up the keyboard */  
  
    ioctl (fileno (stdout), TCSETA, &orig_tty);  
    printf ("\033G@\n");  
    printf ("\033Gc\n");  
    mgclearpln (2,-1,0);  
    /*  
    * Restore the color map  
    */  
    mgicms (0, 32, ColorMap);  
    mgideagp ();  
    system ("/etc/gpc/setcolors 2>>/tmp/code.err");  
    exit(-1);  
}
```

```
*****  
putStrWithBlue
```

Purpose: Puts a given string at a given place of the screen
with a pretty blue rounded box.

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 2/10/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/  
putStrWithBlue(theStr, x)  
    char *theStr; /* The string to put up */  
    float x; /* The normalized x coordinate */  
{  
  
    mgihue (BLACK);  
    mgrbox (x, 0.9120, x+.116 ,0.9320);  
  
    /*  
     * This misbegotten graphics language dumps if  
     * you try to draw a null string.  
     */  
    if(theStr[0]==0)  
        strcpy(theStr,"NULL");  
  
    mgihue (BLUE);  
    mgrbox (x, 0.9120, x + strlen(theStr)*0.01, 0.9320);  
    mgrfc (x, 0.9220, 0.0080);  
    mgrfc (x+strlen(theStr)*0.01, 0.9220, 0.0080);  
    mgihue (YELLOW);  
    mgrgfs (x, 0.9110, 0, theStr);  
}
```



```
/*  
    cleanSlate
```

Purpose: Used by retrieve(), and get_header to initialize the
global variables used by a comp.

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 2/12/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*/  
cleanSlate()  
{  
    ChoiceCounter = 0;  
    ParenCount = 0;  
    Equation = LHS;  
    WhereAmI = CONSEQUENCE;  
    NumberOfIfs = NumberOfEndifs = 0;  
    NumberOfCompares = 0;  
    NumCompVars = 0;  
    FuncParenCount = 0;  
    Comp[0] = 0;  
}
```

```
*****  
get_type
```

Purpose: Used by put_msid, put_local, put_signal to get the
type information.

Designer: Terri Murphy/NASA/JSC/MOD

Programmer: Terri Murphy/NASA/JSC/MOD

Date: 2/12/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/  
get_type(whose_type, the_message)  
char *whose_type;  
char the_message[];  
{  
    int choice;  
  
    color_valid (GET_TYPE, dead_end);  
    inform(the_message, GREEN, 0);  
    choice = menu(RUN);  
  
    if (choice == SHORT)  
    {  
        strcpy (whose_type, "short");  
    }  
    else if (choice == INTEGER)  
    {  
        strcpy (whose_type, "int");  
    }  
    else if (choice == FLOAT)  
    {  
        strcpy (whose_type, "float");  
    }  
    else if (choice == DOUBLE)  
    {  
        strcpy (whose_type, "double");  
    }  
    else if (choice == CHAR)  
    {  
        strcpy (whose_type, "char");  
    }  
}
```

90/06/07
17:07:40

var_check.c

1

VAR_CHECK

Purpose: Var_check checks the variable list to determine whether the variable just input has been previously defined.

Designer: Troy A. Heindel/NASA/JSC/MOD

Programmer: Troy A. Heindel/NASA/JSC/MOD

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

None.

Global Variables

NumCompVars The number of variables in this comp.

Include files

```
#include "code.h"
```

```
var_check (name, local_index, global_index, class)
```

```
char name[]; /* The name of the passed variable */
int *local_index; /* The index into the CompVars struct */
int *global_index; /* The index into the MSIDTable struct */
char class[]; /* Either msid, signal, or local */
```

```
{
    int i; /* Counter */
    int defined_globally;
    int defined_locally;
    char msg_string[150];
    char temp[100];
```

```
    /* Assume the worst--undefined variable */
    defined_globally = FALSE;
    defined_locally = FALSE;
    *local_index = -1;
    *global_index = -1;
```

```
    /* Look for the msid in the struct MSIDTable */
    if (strcmp (class, "msid") == 0)
```

```
{
    /* If there is not MSIDTable then we can't look it up */
    if (!(MSIDTable[0].name))
        inform("MSID's will not be verified; the msid table was not found",PURPLE,2);
    else
```

```
{
    /* Try to find the msid in the tag_MSIDTable */
    for (i=0; i < MSIDCount; i++)
```

```
{
    if (strcmp (name, MSIDTable[i].name) == 0)
    {
        defined_globally = TRUE;
        *global_index = i;
        break;
    }
}
```

```
    }
    /*****
    If we did not find the msid in the msid list
    it is undefined and not usable.
    *****/
    if (!(defined_globally))
    {
        *global_index = ERROR;
    }
}
/*****
Look for signals in the struct SignalTable
*****/
else if (strcmp (class, "signal") == 0)
{
    /*****
    Look for the signal in the SignalTable list
    *****/
    for (i=0; i < SignalCount; i++)
    {
        if (strcmp (name, SignalTable[i].name) == 0)
        {
            defined_globally = TRUE;
            *global_index = i;
            break;
        }
    }
}
/*****
Always check to see whether the variable
is defined locally to this comp.
*****/
for (i=0; i<NumCompVars; i++)
{
    if (strcmp (name, CompVars[i].name) == 0)
    {
        /*****
        Return an ERROR if they are trying to use an msid
        as a local, signal or something of the sort
        *****/
        if ((strcmp (class, CompVars[i].class) == 0) || (strcmp(class, "null") == 0))
        {
            *local_index = i;
            defined_locally = TRUE;
            break;
        }
        else /* It is a miss-classing e.g. a local as a signal */
        {
            return (ERROR);
        }
    }
}
return (OK); /* The variable is ok */
}
```

```

/*****
funcCheck
*****/
```

Purpose: funcCheck checks to see if a given unknown token is defined in the function list. Return 1 if true, 0 otherwise.

Designer: Troy A. Heindel/NASA/JSC/MOD

Programmer: Troy A. Heindel/NASA/JSC/MOD

Date: 12/15/88

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

None.

```
*****/
funcCheck (funcName)
    char funcName[];      /* The name of the professed function */
{
    int i;

    /*****
     Try to find the passed funcName in the list of
     user defined functions, return OK if found
     *****/
    for (i=0; i<NumberOfUserFuncs; i++)
    {
        if (strcmp (funcName, UserFuncs[i]) == 0)
            return (OK);
    }

    /*****
     Return an ERROR if it wasn't a function.
     *****/
    return (ERROR);
}
```

90/06/07
17:07:50

window_io.c

1

/******

 window_io.c

external interfaces:

ask()

inform()

displayWA()

clearWA()

*****/

```
/*
ask()
*/
```

Purpose: Prompts the user for some information.

Returns: 0 - no problem.
1 - they just hit return (implies default).

Designer: Troy Heindel / NASA

Programmer: Troy Heindel / NASA

Date: 2/13/89

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by: Troy Heindel

Reasons for Revision: Improve readability.

```
*****
Include files
*****
```

```
*****
#include "code.h"
#include "color.h"
#include <ctype.h>
#include <termio.h>          /* used for setting up the keyboard */

struct termio orig_tty;     /* used for setting up the keyboard */
struct termio raw_tty;      /* used for setting up the keyboard */

ask (question, color, reply)
char *question,             /* The question to ask the user */
    *reply;                 /* The place to put their reply */
{
    char tempReply[250]; /* A temporary holding place for the string */

    /*
     * Inform the user of the message with 0 wait.
     */
    inform(question,color,0);

    /*
     * Make the Prompt window a black void
     */
    mgihue (BLACK);
    mgrbox (0.1573, 0.0810, 0.9990, 0.1580);

    printf ("\033GD\n"); /* Select keyboard input window */
    printf ("\033Gc\n"); /* Turn text cursor on */
    printf ("\033Gb\n"); /* Make text cursor blink */
    printf ("\033[H\n"); /* Place text cursor in home position */
                          /* in keyboard input window */

    /*
     * Now that we're in the right window, return to original
     * tty mode, and flush the input buffer which might have
     * been filled with garbage, or even worse, WEX worms.
     * Clear the window which still would have that trash.
     */
    ioctl (fileno (stdout), TCSETAW, &orig_tty);
    ioctl (fileno (stdout), TCFLSH, 0);
    printf ("\033[2J\n"); /* Clear keyboard input window */

    /*
     * Null out the given string to be on the safe side
     */
    reply[0] = 0;

    /*
     * Get a string from the user.
     * Copy tempReply into reply only sizeof chars or less.
     */
    gets(reply);

    /*

```

90/06/07
17:07:50

window_io.c

3

```
* Now that we have our input, go back to raw tty mode.
*****/
printf ("\033[2J\n"); /* Clear the window */
ioctl (fileno (stdout), TCSETAW, &raw_tty);
printf ("\033Gac\n"); /* Turn text cursor off */
printf ("\033Gab\n"); /* Make text cursor stop blinking */

/*
 * Redraw the Prompt & Message windows in blue
 *****/
mgihue (BLUE);
mgrbox (0.1573, 0.0810, 0.9990, 0.1580);
mgrbox (0.1573, 0.0010, 0.9990, 0.0790);

fflush(stdout);

/*
 * If they just hit return, return DEFAULT, else OK.
 *****/
if(!reply[0])
    return DEFAULT;
else return OK;
```



```
/******  
    inform()  
*****
```

Purpose: inform() writes a text string to the bottom window.

Designer: Troy A. Heindel/NASA/JSC/MOD

Programmer: Troy A. Heindel/NASA/JSC/MOD

Date: 2/13/89

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

```
*****
```

```
inform (message, color, pauseTime)
```

```
    char message[]; /* The string to be displayed */
```

```
    int  color,      /* The color to display it in */
```

```
    pauseTime; /* The sleep in seconds before erasing */
```

```
(
```

```
/*
```

```
 * Choose the Message window, set the color, draw a box,
```

```
 * place a cursor
```

```
*/
```

```
printf ("\033GG\n"); /* Select Message/Prompt window */
```

```
printf ("\033[2J\n"); /* Clear the window */
```

```
mgihue (color);
```

```
mgrbox (0.1573, 0.0010, 0.9990, 0.0790);
```

```
printf ("\033[H\n"); /* Place (invisible) cursor in home position */
```

```
mgihue(WHITE);
```

```
printf ("%s", message); /* Write message to window */
```

```
fflush(stdout); /* Don't wait to exceed stdout buf size to display */
```

```
/*
```

```
 * Pause for pauseTime, then clear the window.
```

```
*/
```

```
if (pauseTime)
```

```
{
```

```
    sleep(pauseTime);
```

```
    printf ("\033[2J\n"); /* Clear the window */
```

```
    mgihue (BLUE); /* Redraw it in blue */
```

```
    mgrbox (0.1573, 0.0010, 0.9990, 0.0790);
```

```
}
```

```
)
```

```
/******  
displayWA()
```

Purpose: Displays a string in the Work Area window.

Designer: Troy A. Heindel/NASA/JSC/MOD

Programmer: Troy A. Heindel/NASA/JSC/MOD

Date: 2/13/89

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

*****/

```
displayWA(strToDisplay)  
char strToDisplay[];
```

```
{  
    /*  
    * Choose the Work Area window and place the cursor  
    * in its home position.  
    */  
    printf ("\033GC\n");  
    printf ("\033[H\n");
```

```
    /*  
    * Write text to the window  
    */  
    printf ("%s", strToDisplay);  
    fflush(stdout);  
}
```

```
clearWA()
```

```
{  
    /*  
    * Choose and clear the Work Area window  
    */  
    printf ("\033GC\n");  
    printf ("\033[2J\n");  
}
```

90/06/07
13:12:36

archive.c

1

```
*****<----->*****
*
* FILE NAME:    archive.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which are used to backup and restore Comp
*   Groups.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   archive()      - allows the user to backup and restore groups
*   get_arc_func() - determines if the user wants to read, write, or list the device
*
*****<----->*****/
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <X11/Intrinsic.h>
#include <Xm/Text.h>
#include <Xm/MessageB.h>
#include "code.h"
#include "widgets.h"
```

90/06/07
13:12:36

archive.c

2

```

/*****<----->*****/
*
* MODULE NAME:  archive( void )
*
*
* MODULE FUNCTION:
*
* Function determines if the user wants to read, write, or list the backup device.
* Based on the user's desired action, this function accesses the necessary files
* and the backup device.  Display_file() is called to show the user the results
* of the backup/restore operation.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
*                   Data Systems Department
*                   Automation and Data Systems Division
*                   Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/
```

```
void archive()
{
    char    abs_group_name[PATH_LEN],    /* Name with full path */
            dev_str[20],
            group[20],                   /* Name read from GroupNames file */
            grp_name[20],                /* Name without path */
            message[200],                /* A place to build string messages */
            sys_cmd[500],                /* Used with system calls */
            workDir[PATH_LEN];           /* A string to hold current work dir. */

    int     i,j,                         /* counters */
            found,
            rc;                           /* return code from function call */

    FILE     *list_file,
             *tmpfile,
             *grpfile;

    /*
     * Get the working directory, return to calling routine if unsuccessful.
     */

    if (!(getwd(workDir)))
    {
        sprintf ( message, "Backup could not get the working directory" );
        user_ack( message, HELP_U_ACK );
        return;
    }

    /*
     * Setup the default device string.
     */

    switch ( Device )
    {
        case DEV1    : strcpy( dev_str, "/dev/rflp" );
                       break;
        case DEV2    : strcpy( dev_str, "/dev/rctp" );
                       break;
        case DEV3    : strcpy( dev_str, "/dev/rst8" );
                       break;
    }

    sprintf( message, "Do you want to Write, Read, or List the device? (%s)", dev_str);
    rc = get_arc_funct( message );
}
```

90/06/07
13:12:36

archive.c

3

```
if ( rc == ARCH_CANCEL )
    return;

/*
 * User wants to write the files to the device.
 */

if ( rc == ARCH_WRITE )
{
    user_ack("Insert the media, CONTINUE when inserted", HELP_U_ACK);

    /*
     * Get the name of the group to copy to the device.
     */

    rc = get_name(NumOfGroups, "Group", GroupName, &GroupNumber, &Disposition);

    if ( rc == ABORT )
        return;

    if ( rc == ERROR )
    {
        user_ack("Error during file selection, nothing will be written to device", HELP_U_ACK);
        return;
    }

    /*
     * Write files to device.
     */

    list_file = openFile( "/tmp/code.lst", "w", "archive" );
    if ( list_file == NULL )
        return;
    else
    {
        fprintf( list_file, "\n\tLISTING OF FILES WRITTEN TO DEVICE (%s)", dev_str );
        fprintf( list_file, "\n\t-----\n\n\n" );
        fclose ( list_file );
    }

    sprintf( message, "Please wait, writing files to device (%s)...", dev_str );
    inform( message );
    sprintf( sys_cmd, "tar cvf %s %s/%s/*.h %s/%s/*.v %s/%s.dat ", dev_str,
                CodeGroups, GroupName,
                CodeGroups, GroupName,
                AMSupport, GroupName );
    strcat( sys_cmd, ">>/tmp/code.lst 2>>/tmp/code.lst" );

    rc = system ( sys_cmd );
    clear_inform();

    if ( rc != ERROR )
        display_file( LIST, "/tmp/code.lst" );
    else
        user_ack("Backup unsuccessful", HELP_U_ACK);

    system( "rm -f /tmp/code.lst 2>>/tmp/code.err" );
}

/*
 * User wants to read the device to disk.
 */

else if ( rc == ARCH_READ )
{
    /*
     * If they want to read a device, let's first find
     * out what group is on the device.
     */

    sprintf( message, "Checking for CODE files on device (%s)...", dev_str );
    inform ( message );
    sprintf( message, "tar tf %s >/tmp/code.lst 2>>/tmp/code.err", dev_str );
    system ( message );
    clear_inform();
}
```

```
if (! (tmpfile = openFile("/tmp/code.lst", "r", "archive")))
    return;

fscanf (tmpfile, "%s", abs_group_name);
fclose (tmpfile);
system ("rm -f /tmp/code.lst 2>>/tmp/code.err");

/*
 * Since what we have is the full path name
 * and we just want the group name, let's shorten
 * it up.
 */

i = strlen( abs_group_name );

for ( i--, j=0; j < 2; i-- )
    if ( abs_group_name[i] == '/' )
    {
        abs_group_name[i] = NULL;
        j++;
    }

for ( i+=2, j=0; abs_group_name[i] != NULL; i++, j++ )
    grp_name[j] = abs_group_name[i];
grp_name[j] = '\0';

/*
 * Let's open the GroupNames file, and see if the
 * group already exists.
 */

if (! (grpfile = openFile (GroupNamesFile, "r", "archive")))
    return;

found = FALSE;
while (fscanf (grpfile, "%s", group) != EOF)
{
    if (strcmp(grp_name, group) == 0)
    {
        found = TRUE;

        /*
         * If the group already exists, let the user know
         * about it and see if they want to proceed.
         */

        if(! ask("This group already exists, do you want to clobber it?"))
            return;
        break;
    }
}

fclose (grpfile);

/*
 * Let's read the device finally
 */

sprintf( message, "Please wait, reading device (%s) ...", dev_str );
inform( message );

list_file = openFile( "/tmp/code.lst", "w", "archive" );
if ( list_file == NULL )
    return;
else
{
    fprintf( list_file, "\n\tLISTING OF FILES READ FROM DEVICE (%s)", dev_str );
    fprintf( list_file, "\n\t-----\n\n\n" );
    fclose ( list_file );
}

chdir (CodeGroups);
sprintf(sys_cmd, "tar xvf %s %s/%s >>/tmp/code.lst 2>>/tmp/code.lst",
        dev_str, CodeGroups, grp_name);
system (sys_cmd);

chdir (AMSupport);
sprintf(sys_cmd, "tar xvf %s %s/%s.dat >>/tmp/code.lst 2>>/tmp/code.lst",
```

90/06/07
13:12:36

archive.c

5

```
        dev_str, AMSupport, grp_name);
system (sys_cmd);

clear_inform();
display_file( LIST, "/tmp/code.lst" );
system ( "rm -f /tmp/code.lst" );

/*
 * If the group doesn't exist, then add it to
 * GroupNames file.
 */

if(!found)
{
    strcpy(GroupInfo[NumOfGroups].name, grp_name);
    GroupInfo[NumOfGroups++].disposition = 1;
    if (!(grpfile = openFile (GroupNamesFile, "a", "archive")))
        return;

    fprintf(grpfile, "%s 1",grp_name);
    fclose(grpfile);
}

/*
 * User wants to list device.
 */

else if ( rc == ARCH_LIST )
{
    list_file = openFile( "/tmp/code.lst", "w", "archive" );
    if ( list_file == NULL )
        return;
    else
    {
        fprintf( list_file, "\n\tLISTING OF FILES ON DEVICE (%s)", dev_str );
        fprintf( list_file, "\n\t-----\n\n\n" );
        fclose ( list_file );
    }

    user_ack("Insert media into device, OK when inserted", HELP_U_ACK);

    sprintf( sys_cmd,"tar tf %s >>/tmp/code.lst 2>>/tmp/code.lst", dev_str );
    sprintf( message, "Please wait, reading device (%s)...", dev_str );
    inform( message );

    if( system(sys_cmd) == ERROR )
    {
        clear_inform();
        sprintf( message, "Error reading device (%s)", dev_str );
        user_ack( message, HELP_U_ACK );
    }
    else
    {
        clear_inform();
        display_file( LIST, "/tmp/code.lst" );
    }

    system( "rm -f /tmp/code.lst 2>>/tmp/code.err" );
}

chdir(workDir);
}
```

90/06/07
13:12:36

archive.c

6

```

/*****<----->*****/
*
* MODULE NAME:  get_arc_func( prompt )
*
* MODULE FUNCTION:
*
*   Function prompts user for the type of backup/restore operation to perform:
*   read, write, or list.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

int  get_arc_func( prompt )
{
    char *prompt;

    Arg args[1];

    /*
     * Display the archive function popup.
     */

    XtSetArg(args[0],XmNlabelString,XmStringLtoRCreate(prompt,XmSTRING_DEFAULT_CHARSET));
    XtSetValues( lbl_arch, args, 1 );

    if ( SetNum == 1 )
        XtSetArg( args[0], XmNwidth, (250+(4*strlen(prompt))) );
    else
        XtSetArg( args[0], XmNwidth, (300+(8*strlen(prompt))) );
    XtSetValues( frm_arch, args, 1 );

    process_popup( dlg_arch, WAIT );

    /*
     * Return the user's selection.
     */

    return( Get_Funct_Stat );
}
```


90/06/07
13:12:38

cbr_exit.c

1

```
#include <X11/Intrinsic.h>
```

```
extern Widget top;
```

```
/*-----*/
*
* MODULE NAME:  cbr_exit_code( widget, closure, calldata )
*
*
* MODULE FUNCTION:
*
*   Contains the call back function which is used to exit the Comp Builder.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
XtCallbackProc cbr_exit_code ( widget, closure, calldata )
```

```
Widget widget;          /* Set to the widget which initiated this callback
                          * function.
                          */

caddr_t closure,        /* Callback specific data.  This parameter is not
                          * used by this function.
                          */

calldata;               /* Specifies any callback-specific data the widget
                          * needs to pass to the client.  This parameter is
                          * is not used by this function.
                          */

{
    XEvent event;        /* Event structure needed to make the calls to the
                          * XtNextEvent and XtDispatchEvent functions.
                          */

    /*
     * Destroy the root application shell widget and thereby, all subordinate widgets which
     * make up the window and any popup windows used for menus.
     */

    XtDestroyWidget ( top );

    /*
     * Determine if any events have been queued.  These will normally be events which
     * cause the widgets destroy callback to be executed.  Waiting and then processing
     * the events insures that all data structures initialized by the widgets are
     * properly deallocated.
     */

    XtNextEvent ( &event );
    XtDispatchEvent ( &event );

    /*
     * Close the display to deallocate any connections set up by X Windows.  Next
     * exit from the client.
     */

    XCloseDisplay ( XtDisplay(top) );
    exit ( 0 );
}
```

90/06/07
13:12:40

cbr_menu.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Text.h>
#include "code.h"
#include "widgets.h"

extern XtCallbackProc cbr_code_exit();

/*****<----->*****/
*
* MODULE NAME: cbr_code_menu( widget, closure, calldata )
*
*
* MODULE FUNCTION:
*
* Call back function which servers as the main Comp Builder menu.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

XtCallbackProc cbr_code_menu ( widget, closure, calldata )

Widget widget; /* Set to the widget which initiated this callback
                * function.
                */

caddr_t closure, /* Callback specific data. This parameter will be
                  * be set to a value which identifies the selected
                  * command.
                  */

calldata; /* Specifies any callback-specific data the widget
           * needs to pass to the client. This parameter is
           * is not used by this function.
           */

{
    static int newToken; /* This is the token selected by the user.
                          */

    int oldCompNumber,
        oldDisposition,
        oldGroupNumber,
        rc,
        reDraw,
        type;

    char oldCompName [ PATH_LEN ],
        oldGroupName [ PATH_LEN ];

    XmTextPosition ptr;

    newToken = (int)closure;

    /*
     * If the current mode is RUN, determine which token was selected by the user
     * and either change the mode or execute the corresponding routine.
     */

    if ( theMode == RUNmode )
```

90/06/07
13:12:40

cbr_menu.c

2

```
{
    reDraw = TRUE;

    if ( newToken == QUIT )
    {
        if ( NeedToSave )
            if( ask("Do you want to save your latest work before exiting?") )
                save( SHOW_MSG );
        if ( ask("Are you sure you want to exit?") )
            cbr_exit_code( widget, closure, calldata );
        reDraw = FALSE;
    }

    else if ( newToken == BACKUP )
    {
        archive();
        reDraw = FALSE;
    }

    else if ( newToken == COPY )
    {
        if ( Disposition == NO_GROUP )
            user_ack("No group selected; nothing to copy", HELP_U_ACK);
        else
        {
            if ( NeedToSave )
                save( NO_SHOW );
            copy();
        }
        displayWA( Comp );
    }

    else if ( newToken == EDIT )
    {
        if ( Disposition == NO_GROUP )
            user_ack("No group selected; nothing to edit", HELP_U_ACK);
        else
            edit();
        reDraw = FALSE;
    }

    else if ( newToken == SAVE )
    {
        if ( Disposition == NO_GROUP )
            user_ack("No group selected; nothing to save", HELP_U_ACK);
        else
            save( SHOW_MSG );
        reDraw = FALSE;
    }

    /*
     * User wants to select fonts.
     */

    else if ( newToken == FONTS )
    {
        PopupHelp      = HELP_FONTS;
        SaveWorkAFont = WorkA;
        set_font_tgl();
        process_popup( dlg_font, WAIT );
    }

    /*
     * User wants to select fonts.
     */

    else if ( newToken == DEVICE )
    {
        PopupHelp      = HELP_DEVICE;
        SaveDevice     = Device;
        set_device_tgl();
        process_popup( dlg_device, WAIT );
    }

    /*
     * User wants to print files.
     */
}
```

90/06/07
13:12:40

cbr_menu.c

3

```
else if ( newToken == HARDCOPY )
{
    if ( Disposition == NO_GROUP )
        user_ack("No group selected; nothing to print", HELP_U_ACK);
    else
    {
        if ( NeedToSave )
            save( NO_SHOW );
        hardcopy();
    }
    reDraw = FALSE;
}

else if ( newToken == INSTALL )
{
    if ( Disposition == NO_GROUP )
        user_ack("No group selected; nothing to install", HELP_U_ACK);
    else
    {
        if ( NeedToSave )
            save( NO_SHOW );
        select_cursor( Clock_Cursor );
        install();
    }
}

else if ( newToken == REMOVE )
{
    if ( NeedToSave )
        if( ask("Do you want to save your latest work first?") )
            save( SHOW_MSG );

    remove();

    reDraw = FALSE;
}

/*
 * The token selected is DELETE, check to make sure there is a valid Comp
 * in the work area.
 */

else if ( newToken == DELETE )
{
    if ( Disposition == NO_GROUP )
    {
        user_ack("No group selected; nothing to delete", HELP_U_ACK);
        return;
    }

    /*
     * If there are valid tokens, then delete the last one.
     */

    delete();

    /*
     * If we are down to the last token, then start the comp over again by
     * getting the header.
     */

    if ( ChoiceCounter < 1 )
        get_header();

    displayWA( Comp );
    ptr = XmTextGetLastPosition( txt_worka );
    XmTextShowPosition( txt_worka, ptr );
}

/*
 * The token selected is RETRIEVE, give the user a chance to save the previous
 * work.
 */

else if ( newToken == RETRIEVE )
{
    if ( NeedToSave )
```

90/06/07
13:12:40

cbr_menu.c

4

```
if( ask("Do you want to save your latest work first?") )
    save( SHOW_MSG );

/*
 * Keep the old name/number around in case the user aborts from the retrieve.
 * Get a group name from the user to retrieve.
 */

strcpy( oldGroupName, GroupName );
oldGroupNumber = GroupNumber;
oldDisposition = Disposition;
rc = get_name( NumOfGroups, "Group", GroupName, &GroupNumber, &Disposition );

if ( rc != ERROR )
{
    /*
     * Keep the comp name and number around in 'old' variables in case
     * the user aborts the get_name. If we do not have an error getting
     * the name, then retrieve the chosen comp.
     */

    oldCompNumber = CompNumber;
    strcpy( oldCompName, CompName );
    strcpy( CompName, " " );
    put_status();

    rc = get_name( NumOfComps, "Comp", CompName, &CompNumber, &Disposition );

    if ( rc != ERROR )
    {
        select_cursor( Clock_Cursor );
        retrieve();
        displayWA( Comp );
        select_cursor( Shuttle_Cursor );
    }
    else
    {
        strcpy( GroupName, oldGroupName );
        strcpy( CompName, oldCompName );
        Disposition = oldDisposition;
        GroupNumber = oldGroupNumber;
        CompNumber = oldCompNumber;
        reDraw = FALSE;

        if ( Disposition != NO_GROUP )
            readCompNames();
    }
}
else
{
    strcpy( GroupName, oldGroupName );
    Disposition = oldDisposition;
    GroupNumber = oldGroupNumber;
    reDraw = FALSE;
}

}

/*
 * The token selected is CREATE, give the user a chance to save the previous
 * work.
 */

else if ( newToken == CREATE )
{
    if ( NeedToSave )
        if( ask("Do you want to save your latest work first?") )
            save( SHOW_MSG );

    /*
     * If there are zero groups then they must want a new group since
     * comps must be associated with a group, otherwise, determine if
     * the user wants a new Group or a new Comp.
     */

    if ( NumOfGroups < 1 )
        type = GROUP;
```

90/06/07
13:12:40

cbr_menu.c

5

```
else
{
    type = get_type_gc( "Would you like a new Group or Comp?" );
    if ( type == ABORT )
    {
        put_status();
        return;
    }
}

/*
 * User wants a new Group.
 */

if ( type == GROUP )
{
    if ( new("Group") != DEFAULT )
        if ( new("Comp") != DEFAULT )
            get_header();
}

/*
 * User wants a new Comp.
 */

else if ( type == COMP )
{
    /*
     * Get the name of the group to put the new comp into.
     */

    rc = get_name( NumOfGroups, "Group", GroupName, &GroupNumber, &Disposition);
    if ( rc != ERROR )
    {
        put_status();

        /*
         * Read the existing comps associated with this group.
         */

        readCompNames();
    }

    /*
     * Create the new comp.
     */

    if ( new("Comp") != DEFAULT )
        get_header();
}

/*
 * Clear the "NEW" widget.
 */

displayWA( Comp );
}

/*
 * The token selected is 1 SHOT or CYCLIC.
 */

else if ((newToken == TGL_1SHOT) || (newToken == TGL_CYCLE))
{
    if ( newToken == TGL_1SHOT )
        set_cycle_mode( ONE_SHOT );
    if ( newToken == TGL_CYCLE )
        set_cycle_mode( CYCLIC );
    put_status();
    reDraw = FALSE;
}

/*
 * The token selected is LIST, call the correct List routine.
 */

else if ( newToken == LIST_MSID )
```

90/06/07
13:12:40

cbr_menu.c

6

```
token_help( MSID, LIST, widget );
else if ( newToken == LIST_SIGNAL )
    token_help( SIGNAL, LIST, widget );
else if ( newToken == LIST_FUNCT )
    token_help( FUNCTION, LIST, widget );
else if ( newToken == LIST_CYCLE )
    display_file( LIST, "./CYCLE_MODES" );

/*
 * The token selected is HELP, change to the HELP mode, display a message to
 * the user that he/she is in HELP mode.
 */

else if ( newToken == HELP_TOKENS )
{
    theMode = HELP;

    /*
     * Set all widgets to sensitive.
     */

    PopupHelp = HELP_HLP;
    all_valid();
    select_cursor( Help_Cursor );
    process_popup( dlg_help, NO_WAIT );
    reDraw = FALSE;
}

else if ( newToken == HELP_MANUAL )
{
    token_help( TOC, HELP, widget );
    reDraw = FALSE;
}

/*
 * The token selected must be a Comp element. If the token's function pointer
 * is NULL, then the current function is not yet implemented, notify the user.
 * If the function pointer is not NULL, execute the token's action function.
 */

else
{
    if ( tokens[ newToken ].function == NULL )
    {
        user_ack("This function is not yet implemented", HELP_U_ACK);
        reDraw = FALSE;
    }
    else
    {
        if ( tokens[ newToken ].id )
            (*(tokens[ newToken ].function)) ( newToken );
        else
            (*(tokens[ newToken ].function)) ();
    }
}

/*
 * Update the screen, change the valid tokens list.
 */

if ( reDraw )
{
    color_valid( PrevChoice[ChoiceCounter-1], newToken );
    put_status();
}

/*
 * User selected Token Help then a token, display the help
 * for the selected token.
 */

else if ( theMode == HELP )
{
    theMode = RUNmode;
    select_cursor( Shuttle_Cursor );
}
```

90/06/07
13:12:40

cbr_menu.c

7

```
XtUnmanageChild( dlg_help );  
token_help( newToken, HELP, widget );
```

```
/*  
 * Restore the list of valid tokens.  
 */
```

```
color_valid( IGNORE, IGNORE );  
return;  
}
```


90/06/07
13:12:42

cbr_popups.c

1

```
/*-----*/
*
* FILE NAME:      cbr_popups.c
*
* FILE FUNCTION:
*
*   Contains the callback routines for all popup buttons.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* FILE MODULES:
*
*   cbr_clear_get_name()  - clears the get_name() popup
*   cbr_clear_popup()    - callback to clear popups
*   cbr_show_help()      - callback to clear help popup
*
*-----*/
```

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
#include "widgets.h"
```

90/06/07
13:12:42

cbr_popups.c

2

```
/*-----*/
*
* MODULE NAME:  cbr_clear_get_name( widget, closure, calldata )
*
*
* MODULE FUNCTION:
*
*   Callback routine to clear the get group/comp name popup.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/

XtCallbackProc  cbr_clear_get_name( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
           calldata;

{
    if ( (int) closure == GN_HELP )
        token_help( PopupHelp, HELP, NULL );
    else
    {
        Get_Name_Stat = (int) closure;
        PopupStat[ ActivePopup ] = TRUE;
        XtUnmanageChild( dlg_gname );
    }
}
```

90/06/07
13:12:42

cbr_popups.c

3

```

/*****<----->*****/
*
* MODULE NAME:  cbr_clear_popup( widget, closure, calldata)
*
*
* MODULE FUNCTION:
*
*   Callback routine to handle all the popup buttons.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

XtCallbackProc  cbr_clear_popup( widget, closure, calldata )

Widget  widget;
caddr_t closure,
        calldata;

(
/*
 * Determine which button the user pushed, act accordingly.
 */
switch ( (int) closure )
{
    case ARCH_READ   :
    case ARCH_LIST   :
    case ARCH_WRITE   :
    case ARCH_CANCEL: Get_Funct_Stat = (int) closure;
                     XtUnmanageChild( dlg_arch );
                     break;

    case ASK_NO      : Ask_Resp = FALSE;
                     XtUnmanageChild( dlg_ask );
                     break;

    case ASK_YES     : Ask_Resp = TRUE;
                     XtUnmanageChild( dlg_ask );
                     break;

    case FILE_OK     : XtUnmanageChild( dlg_file );
                     break;

    case FONT_CANCEL: WorkA = SaveWorkAFont;
                     XtUnmanageChild( dlg_font );
                     break;

    case FONT_OK     : set_size();
                     XtUnmanageChild( dlg_font );
                     break;

    case GC_COMP     : XtUnmanageChild( dlg_gc );
                     Get_GC_Stat = COMP;
                     break;

    case GC_GROUP    : XtUnmanageChild( dlg_gc );
                     Get_GC_Stat = GROUP;
                     break;

    case GC_CANCEL   : XtUnmanageChild( dlg_gc );
                     Get_GC_Stat = ABORT;
                     break;

    case G_OK        : Get_Str_Stat = ACCEPT;
                     theMode      = RUNmode;
                     break;

    case G_CANCL     : Get_Str_Stat = ABORT;
                     theMode      = RUNmode;
                     break;
}
```

```
case HELLO      : XtUnmanageChild( dlg_hello );
                  break;
case HELP_CANCEL : XtUnmanageChild( dlg_help );
                  color_valid( IGNORE, IGNORE );
                  theMode = RUNmode;
                  select_cursor( Shuttle_Cursor );
                  break;

case HELP_CLOSE : XtUnmanageChild( dlg_help_txt );
                  break;
case INFORM      : break;
case SHORT       : break;
case INTEGER     : break;
case FLOAT       : break;
case DOUBLE      : break;
case CHAR        : XtUnmanageChild( dlg_type );
                  Get_Type_Stat = (int) closure;
                  break;
case TGL_DEV1    : Device = DEV1;
                  set_device_tgl();
                  break;
case TGL_DEV2    : Device = DEV2;
                  set_device_tgl();
                  break;
case TGL_DEV3    : Device = DEV3;
                  set_device_tgl();
                  break;
case T_CANCEL    : XtUnmanageChild( dlg_type );
                  Get_Type_Stat = ABORT;
                  break;
case TGL_WA_DE   : WorkA = DEFAULT;
                  set_font_tgl();
                  break;
case TGL_WA_O1   : WorkA = OPTION1_FONT;
                  set_font_tgl();
                  break;
case TGL_WA_O2   : WorkA = OPTION2_FONT;
                  set_font_tgl();
                  break;
case USER_ACK   : XtUnmanageChild( dlg_ack );
                  break;

case DEVICE_CANCEL : XtUnmanageChild( dlg_device );
                  Device = SaveDevice;
                  break;
case DEVICE_OK    : XtUnmanageChild( dlg_device );
                  break;
}
```

```
PopupStat[ ActivePopup ] = TRUE;
```

```
}
```

90/06/07
13:12:42

cbr_popups.c

5

```
/*-----*/
*
* MODULE NAME:  cbr_show_help( widget, closure, calldata)
*
*
* MODULE FUNCTION:
*
*   Callback routine to handle requests for popup help.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
XtCallbackProc  cbr_show_help( widget, closure, calldata )
```

```
Widget  widget;
caddr_t closure,
        calldata;
```

```
{
/*
 * Determine which popup wants help, display the corresponding help text.
 */
```

```
switch ( (int) closure )
{
    case FILE_HELP  : process_popup( dlg_help_txt, WAIT );
                      break;
    case GC_HELP    : token_help( HELP_GC, HELP, NULL );
                      break;
    case HELP_HELP  : token_help( PopupHelp, HELP, NULL );
                      select_cursor( Help_Cursor );
                      break;
    case ARCH_HELP  :
    case T_HELP     :
    case ASK_HELP   :
    case DEVICE_HELP:
    case FONT_HELP  :
    case G_HELP     :
    case USER_HELP  : token_help( PopupHelp, HELP, NULL );
                      break;
}
```

```
}
```

90/06/07
13:12:44

cbr_popups.h

1

```
/*----->*****  
*  
* FILE NAME:      cbr_popups.h  
*  
*  
* FILE FUNCTION:  
*  
* This file contains the callback function prototypes and the callback list  
* structures used to construct the popups.  
*  
*  
* SPECIFICATION DOCUMENTS:  
*  
* /code/specs/code  
*  
*  
* FILE MODULES:  
*  
* N/A  
*  
*****<----->*/  
  
/*  
* Function prototypes for callback routines.  
*/  
  
XtCallbackProc  cbr_code_menu(),  
                cbr_clear_get_name(),  
                cbr_clear_popup(),  
                cbr_exit_code(),  
                cbr_show_help();  
  
/*  
* Callback list structures.  
*/  
  
XtCallbackRec clear_get_name[] = {  
    { (XtCallbackProc)cbr_clear_get_name, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,                (caddr_t)NULL }  
};  
  
XtCallbackRec clear_popup[] = {  
    { (XtCallbackProc)cbr_clear_popup, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,            (caddr_t)NULL }  
};  
  
XtCallbackRec clear_popup1[] = {  
    { (XtCallbackProc)cbr_clear_popup, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,            (caddr_t)NULL }  
};  
  
XtCallbackRec  code_menu[] = {  
    { (XtCallbackProc)cbr_code_menu, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,          (caddr_t)NULL }  
};  
  
XtCallbackRec  exit_code[] = {  
    { (XtCallbackProc)cbr_exit_code, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,          (caddr_t)NULL }  
};  
  
XtCallbackRec gc_comp[] = {  
    { (XtCallbackProc)cbr_clear_popup, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,            (caddr_t)NULL }  
};  
  
XtCallbackRec gc_group[] = {  
    { (XtCallbackProc)cbr_clear_popup, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,            (caddr_t)NULL }  
};  
  
XtCallbackRec show_help[] = {  
    { (XtCallbackProc)cbr_show_help, (caddr_t)NULL },  
    { (XtCallbackProc)NULL,          (caddr_t)NULL }  
};
```

90/06/07
13:12:46

code.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
#include "tokens.h"
```

```
/*-----*/
*
* MODULE NAME: main( argc, argv )
*
*
* MODULE FUNCTION:
*
*   Main function of CODE.  Inititalizes the X Windows system, builds all
*   application widgets, initializes all files and paths, and then begins
*   main menu.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
main( argc, argv )
int    argc;
char   **argv;
{
    ActivePopup = -1;

    /*
     * Initialize the X Windows system and build all widgets and popups.
     */

    init_graphics( argc, argv );

    /*
     * Initialize any files and paths required by CODE.
     */

    if ( init_code() == ERROR )
        cleanExit();

    /*
     * Enter the normal Xtoolkit main loop, which coordinates processing of the various
     * widget events.  This loop will terminate normally when the user selects the "Quit"
     * command, which in turn causes the exit_code callback routine to be executed.
     */

    XtMainLoop();

    /*
     * Program should exit via cbr_exit(), but let's make lint happy.
     */

    return( OK );
}
```

90/06/07
13:12:48

code.h

1

```

/*****<----->*****/
*
* FILE NAME:    code.h
*
*
* FILE FUNCTION:
*
*   This file contains the declaration of the Comp Builder global variables and
*   defines many of the data structures which are maintained.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   N/A
*
*****/
```

```
#include "code_const.h"
#include "code_extern.h"
```

```
/*
 * Structure definitions
 */
```

```
struct comp_info_struct
{
    char    name[ MAX_NAME_LEN ],
           purpose[ 80 ];
    int     cycle_mode,
           disposition,
           noise_filter,
           on_off,
           rate;
};
struct comp_info_struct CompInfo[ MAX_COMPS+1 ];
```

```
struct group_info_struct
{
    char    name[ MAX_NAME_LEN ];
    int     disposition;
};
struct group_info_struct GroupInfo[ MAX_GROUPS ];
```

```
struct msid_tbl_struct
{
    char    name[ MSID_NAME_LEN ],
           type[ TYPE_LEN ],
           nomenclature[ NOMEN_LEN ];
};
struct msid_tbl_struct MSIDTable[ MAX_NUM_MSIDS ];
```

```
struct sig_tbl_struct
{
    char    name[ SIGNAL_NAME_LEN ],
           type[ TYPE_LEN ];
    int     putter_count;
    char    putter[ MAX_SETTERS ][ MAX_NAME_LEN + MAX_NAME_LEN + 2 ];
    int     getter_count;
    char    getter[ MAX_SETTERS ][ MAX_NAME_LEN + MAX_NAME_LEN + 2 ];
    char    nomenclature[ NOMEN_LEN ];
};
struct sig_tbl_struct SignalTable[ MAX_NUM_SIGNALS ];
```

C-4


```

struct token_tag
{
    char    name[ MAX_TOKEN_NAME ];
    int     help,
            (*function)(),
            id;
};

struct var_struct
{
    char    class      [ TYPE_LEN ],
            name       [ MAX_VAR_LEN ],
            type       [ TYPE_LEN ],
            nomenclature[ NOMEN_LEN ];
    int     occurrence,
            put_or_get;
    float   hi1_limit,
            hi2_limit,
            lo1_limit,
            lo2_limit;
};

struct var_struct CompVars [ MAX_COMP_VARS ],
                  GroupVars[ MAX_GROUP_VARS ];

/*
 * Globals
 */

char    AM
        AMGroups      [ PATH_LEN ],
        AMSupport     [ PATH_LEN ],
        Code          [ PATH_LEN ],
        CodeDocs      [ PATH_LEN ],
        CodeGroups    [ PATH_LEN ],
        Comp          [ MAX_COMP_LEN ],
        CompareType    [ MAX_COMPARES ][ TYPE_LEN ],
        CompName       [ MAX_NAME_LEN ],
        FunctionList   [ MAX_NUM_FUNCS ][ FUNC_NAME_LEN ],
        GroupName      [ MAX_NAME_LEN ],
        GroupNamesFile [ PATH_LEN ],
        Help_Txt_Str   [ 250 ],
        List_Txt_Str   [ 250 ],
        MSIDTbl        [ PATH_LEN ],
        RTDS           [ PATH_LEN ],
        SignalTbl      [ PATH_LEN ],
        UserFuncs      [ MAX_NUM_FUNCS ][ FUNC_NAME_LEN ],
        UserFuncsLib   [ PATH_LEN ];

int     ActivePopup,
        Ask_Resp,
        ChoiceCounter, /* An index into the PrevChoice array of tokens */
        ColorMap[32], /* array to store and restore the color map */
        CompNumber, /* The index of the comp in the group_info structure */
        Cycle_Mode, /* Cycle Selection Mode */
        Disposition,
        Equation, /* LHS or RHS */
        FunctionCount, /* The number of functions used in a comp */
        FunctionArguments[ MAX_NUM_FUNCS ], /* The number or arguments needed to satisfy the function */
        FunctionArgsDef[ MAX_NUM_FUNCS ], /* The number or arguments entered thus far */
        FunctionCurrent, /* Function currently trying to satisfy arguments for */
        FuncParenCount, /* The paren count for function argument definitions */
        Get_Funct_Stat,
        Get_GC_Stat,
        Get_Name_Stat,
        Get_Str_Stat,
        Get_Type_Stat,
        GroupNumber, /* The index of the group in the group_list structure */
        LikelyNextChoice, /* Token to position the mouse over; menu(), next_inputs() */
        MSIDCount, /* The number of msids read from the tag_msid_tbl */
        NeedToSave, /* TRUE means we have made changes to the WA comp */
        NestedElseCheck[ MAX_NESTED_IF ], /* Used in next_inputs to determine if else is valid */
        NumCompVars, /* Used in save, retrieve, var_check. The number of variables used in a comp. */
        NumGroupVars, /* Used: save, retrieve. Number variables used in a group. */
        NumOfComps, /* The number of comps in the active group */

```

90/06/07
13:12:48

code.h

3

```
NumOfGroups,          /* The number of groups */
NumberOfIfs,          /* The number of ifs in the work area comp */
NumberOfEndifs,       /* The number of endifs in the work area comp */
NumberOfCompares,     /* The number of equal signs in a comp, both EQ, and EQEQ */
NumberOfUserFuncs,    /* the number of user funcs read into init_code.c */
PopupHelp,
PopupStat[5],
ParenCount,           /* The number of parenthesis */
PrevChoice[MAX_TOKENS], /* An integer history of the tokens in a comp */
SetNum,              /* */
SignalCount,         /* The number of signal read from the signal table */
theMode,
TotValPts,           /* The number tokens which are valid; menu(), next_inputs() */
ValidPoints[NUMBER_OF_OPTIONS], /* The list of valid tokens; menu() and next_inputs() */
WaitCursor,
WhereAmI;            /* Either PREMISE or CONSEQUENCE */

int    Device,
      SaveDevice,
      SaveWorkAFont,
      WorkA,

      BT_Height,
      EL_Height,
      ST_Height,
      WA_Height,

      BT_Width,
      EL_Width,
      FM_Width;

char    BT_Font[30],
      EL_Font[30],
      *EL_IColor,
      Font_Default[30],
      *Font_Option1,
      *Font_Option2,
      *ST_Font,
      *WA_Font;

Cursor  Clock_Cursor,
      Help_Cursor,
      Shuttle_Cursor;

Display *display;

Pixel   insensitive_color,
      sensitive_color;

XmFontList Fnt_List,
      Fnt_List_Btn;
```

90/06/07
13:12:50

code_const.h

1

```
*****<----->*****
*
* FILE NAME:      code_const.h
*
*
* FILE FUNCTION:
*
*   This file contains all the Comp Builder program constants.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   N/A
*
*****<----->*****/
```

```
#define NO_SCROLL      0
#define SCROLL_BARS    1
```

```
/*
 * Constants
 */
```

```
#ifndef FALSE
#define FALSE          0
#endif FALSE
```

```
#ifndef TRUE
#define TRUE           1
#endif TRUE
```

```
#ifndef NULL
#define NULL           0
#endif NULL
```

```
#define ABORT          -1
#define ACCEPT          0
#define CMD_LEN        150
#define COMP           3
#define COMPLETE        1
#define CONSEQUENCE     1
#define CYCLIC          1
#define DEFAULT         1
#define DEV1            1
#define DEV2            2
#define DEV3            3
#define ERROR          -1
#define FIRST_SIGNAL    "SIGNAL_ZERO"
#define FIRST_TIME      1
#define FUNC_NAME_LEN   11
#define GET             1
#define GROUP           2
#define HELP_TEXT_NO    0
#define HELP_TEXT_YES   1
#define IGNORE          -2
#define INCOMPLETE      0
#define INSTALLED       2
#define LARGE           68
#define LHS             0
#define MAX_COMMENT_LEN 250
#define MAX_COMP_LEN    5000 /* 20000 */
#define MAX_COMP_VARS   50
#define MAX_COMPARES    100
#define MAX_COMPS       50 /* was 50 */
#define MAX_FUNC_PARAMS 20
#define MAX_GROUP_VARS  500
#define MAX_GROUPS      100
#define MAX_MSG_LEN     300
#define MAX_NAMES       50
#define MAX_NAME_LEN    14
```

90/06/07
13:12:50

code_const.h

2

```
#define MAX_NESTED_IF      12
#define MAX_NUM_FUNCS      100
#define MAX_NUM_MSIDS      4000
#define MAX_NUM_SIGNALS    300 /* was 800 */
#define MAX_SETTERS        50
#define MAX_STR_LEN        120
#define MAX_TOKENS         2000
#define MAX_TOKEN_NAME     15
#define MAX_VAR_LEN        30
#define MEDIUM             67
#define MSID_NAME_LEN      12
#define NO_COLOR           0
#define NO_GROUP           -2
#define NO_SHOW            0
#define NO_WAIT            0
#define NOMEN_LEN          50
#define NONE               0
#define NULLS              ""
#define NUMBER_OF_OPTIONS  68
#define OK                 0
#define ONE_SHOT           2
#define OPTION1_FONT       2
#define OPTION2_FONT       3
#define PATH_LEN           80
#define PREMISE            0
#define PET                2
#define PUT                0
#define READ               0
#define RHS                1
#define RUN                999
#define RUNmode            1
#define SECOND_TIME        0
#define SHOW_MSG           1
#define SIGNAL_NAME_LEN    21
#define SIZE_INDENT        5
#define SMALL              66
#define TOC                0
#define TYPE_LEN           10
#define WAIT               1
#define WORK_AREA          66
#define WRITE              1
```

```
/*
 * Define Popup ids.
 */
```

```
#define ARCH_CANCEL      101
#define ARCH_HELP        102
#define ARCH_LIST        103
#define ARCH_READ        104
#define ARCH_WRITE       105
#define ASK_HELP         106
#define ASK_NO           107
#define ASK_YES          108
```

```
#define DEVICE_CANCEL    150
#define DEVICE_HELP      151
#define DEVICE_OK        152
```

```
#define FILE_OK          109
#define FILE_HELP        110
#define FONT_CANCEL      111
#define FONT_OK          112
#define FONT_HELP        113
#define HELLO            114
#define HELP_CANCEL      115
#define HELP_CLOSE       116
#define HELP_HELP        117
#define HELP_TOKENS      118
#define HELP_MANUAL      119
#define GC_CANCEL        120
#define GC_COMP          121
#define GC_GROUP         122
#define GC_HELP          123
#define GN_HELP          124
#define G_CANCL          125
#define G_HELP           126
#define G_OK             127
```

90/06/07
13:12:50

code_const.h

3

```
#define INFORM          128
#define LIST_CYCLE      129
#define LIST_FUNCT      130
#define LIST_MSID       131
#define LIST_SIGNAL     132
#define TGL_1SHOT       133
#define TGL_CYCLE       134
#define TGL_DEV1        135
#define TGL_DEV2        136
#define TGL_DEV3        137
#define TGL_WA_DE       138
#define TGL_WA_O1       139
#define TGL_WA_O2       140
#define T_CANCL         141
#define T_HELP          142
#define USER_ACK        143
#define USER_HELP       144

/*
 * Define menu selection ids. Do not change the order
 * of the following menu ids without changing the tokens
 * structure in tokens.h.
 */

#define CREATE          1
#define EDIT            2
#define DELETE          3

#define IF              4
#define ELSE            5
#define AND             6
#define BITXOR          7
#define BITAND          8
#define THEN            9
#define END_IF         10
#define OR              11
#define NOT             12
#define BITOR           13

#define MSID            14
#define LOCAL           15
#define STRING          16
#define SIGNAL          17
#define NUMBER          18

#define SHORT           19
#define FLOAT           20
#define CHAR            21
#define INTEGER         22
#define DOUBLE          23

#define EQ              24
#define NE              25
#define LE              26
#define GE              27
#define LT              28
#define GT              29

#define SET             30
#define PRINT           31
#define FUNCTION        32
#define COMMENT         33
#define START           34
#define STOP            35

#define RETRIEVE        36
#define SAVE            37
#define INSTALL         38
#define COPY            39
#define REMOVE          40
#define HARDCOPY        41
#define BACKUP          42
#define QUIT            43

#define HELP            44
#define LIST            45

#define ADD             46
```

90/06/07
13:12:50

code_const.h

4

```
#define SUBTRACT      47
#define MULTIPLY      48
#define DIVIDE        49
#define L_PAREN       50
#define R_PAREN       51
#define COMMA         52
#define PI            53
#define SQRT          54
#define POWER         55
#define EXP           56
#define LOG           57
#define SHIFTL        58
#define SHIFTR        59
#define COS           60
#define SIN           61
#define TAN           62
#define ACOS          63
#define ASIN          64
#define ATAN          65
```

```
#define COLORS        66
#define FONTS         67
#define DEVICE        68
```

```
/*
 * Define Popup Help ID's - these are tied directly to the
 * tokens[] array in tokens.h.
 */
```

```
#define HELP_GC       80
#define HELP_G_SEL    81
#define HELP_C_SEL    82
#define HELP_FONTS    83
#define HELP_U_ACK    84
#define HELP_HLP      85
#define HELP_DEVICE   86
```

90/06/07
13:12:53

code_extern.h

1

```
/*-----*/
*
* FILE NAME:    code_extern.h
*
*
* FILE FUNCTION:
*
*   This file contains most of the Comp Builder function prototypes.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   N/A
*
*
*-----*/
```

```
Dimension  get_height  (),
           get_width  (),
           get_x      (),
           get_y      ();
```

```
FILE       *openFile();
```

```
XImage     *CreateDefaultImage();
```

```
extern struct token_tag tokens[];
```

```
void       all_valid      (),
           archive        (),
           arm_toggle     (),
           build_get_name (),
           center_horz    (),
           center_widget  (),
           cleanExit      (),
           cleanSlate     (),
           clear_inform   (),
           color_valid    (),
           disarm_toggle  (),
           display_file   (),
           display_str    (),
           displayWA      (),
           edit           (),
           get_defaults   (),
           hello_screen   (),
           inform         (),
           init_get_name_popup(),
           init_graphics  (),
           LIST_valid     (),
           popup_wait     (),
           process_popup  (),
           select_cursor  (),
           set_btn_defaults(),
           set_cycle_mode (),
           set_defaults   (),
           set_device_tgl (),
           set_font_tgl   (),
           token_help     (),
           user_ack       (),
           unset_widget   (),
           update_SA      (),
           updateWA       ();
```

```
int        load_font      (),
           put_add        (),
           put_and        (),
           put_bitAnd     (),
           put_bitOr      (),
           put_bitXor     (),
           put_comma      (),
```

```
put_comment      (),
put_divide       (),
put_else         (),
put_endif        (),
put_eq           (),
put_func         (),
put_ge           (),
put_gt           (),
put_if           (),
put_l_paren      (),
put_le           (),
put_local        (),
put_lt           (),
put_msld         (),
put_multiply     (),
put_ne           (),
put_not          (),
put_number       (),
put_or           (),
put_pi           (),
put_print        (),
put_r_paren      (),
put_set          (),
put_shiftL       (),
put_shiftR       (),
put_signal       (),
put_start        (),
put_stop         (),
put_string       (),
put_subtract     (),
put_then         (),
readCompNames    (),
readGroupNames   ();
```


90/06/07
13:12:55

copy.c

1

COPY

Purpose: Copy copies a comp or a group into a new comp or group.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/20/89

Version: 2.0

Project: CODE (Comp Development Environment)

External Interfaces

```
new()          -- creates a new comp or group
putStrWithBlue() -- puts the name on status line in bubble
save()         -- saves the current comp/group to disk
switch_name()  -- changes the comp name in its header
*****/
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
```

```
copy ()
{
    char    reply[200],          /* a temp string */
            oldName[MAX_NAME_LEN], /* a place to hold the new name */
            pathOldGroup[PATH_LEN], /* the path for the old group */
            pathNewGroup[PATH_LEN]; /* the path for the new group */

    int     oldNumber,          /* Used to store the number of comps */
            rc;

    /*
     * See if they want to copy a comp or a group
     */

    rc = get_type_gc( "Make copy of this Comp or Group?" );
    if ( rc == ABORT )
        return( ERROR );

    if( rc == GROUP )
    {
        /*
         * First off, let's build some paths to the source
         * groups files, so we can copy them later on into
         * the destination groups files
         */

        sprintf(pathOldGroup,"%s/%s",CodeGroups,GroupName);

        /*
         * Get the name of the comp the user wishes
         * to copy from get_new_name.
         */

        strcpy(oldName, GroupName);
        oldNumber = GroupNumber;
        if (get_new_name("Group", GroupName))
            return( ABORT );

        /*
         * Copy the comps from the old group into the new one.
         * Note that GroupName was modified in get_new_name()!
         */

        sprintf(pathNewGroup,"%s/%s",CodeGroups,GroupName);
        if(mkdir (pathNewGroup, 511))
        {
            user_ack("CODE is unable to make the new directory - copy aborted.", HELP_U_ACK);
            strcpy (GroupName, oldName); /* Undo the damage */
            return (ERROR);
        }

        sprintf(reply,"cp %s/* %s>/tmp/tart 2>>/tmp/code.err",pathOldGroup,pathNewGroup);

        /*
         * If we fail to copy the files we should go back to the
         * original configuration.
         */

        if(system(reply)!=0)
        {
            user_ack("CODE is unable to copy the files into the new directory; copy aborted.", HELP_U_ACK);
            strcpy (GroupName, oldName); /* Undo the damage */
            return (ERROR);
        }

        /*
         * Since we had a successful copy, copy the group
         * info from the old group into the new group, and
         * copy the GroupName into the GroupInfo.
         */

        GroupNumber = NumOfGroups++;
        GroupInfo[GroupNumber] = GroupInfo[oldNumber];
        strcpy(GroupInfo[GroupNumber].name, GroupName);

        /*
         * Paste the new group name up on the status line
         */
    }
}
```

90/06/07
13:12:55

copy.c

3

```
*/  
  
    put_status();  
    save( NO_SHOW );  
}  
  
/*  
 * else they want to copy a COMP  
 */  
  
else if( rc == COMP )  
{  
    /*  
     * Get the name of the comp the user wishes  
     * to copy from from new(), but  
     * remember to keep the old name around for  
     * use in the call to switch_name().  
     */  
    strcpy(oldName, CompName);  
    oldNumber = CompNumber;  
    if (get_new_name("Comp", CompName))  
        return( ABORT );  
  
    /*  
     * Switch the old name which is in the comps header  
     */  
  
    if (switch_name (oldName) == ERROR)  
        user_ack("An error occurred while changing the name.", HELP_U_ACK);  
  
    /*  
     * Copy the old comp info into the new comp info,  
     * copy the CompName into the CompInfo  
     * and increment the NumOfComps.  
     */  
  
    CompNumber = NumOfComps++;  
    CompInfo[CompNumber] = CompInfo[oldNumber];  
    strcpy(CompInfo[CompNumber].name, CompName);  
  
    /*  
     * Paste the group & new comp name up on the screen  
     */  
  
    put_status();  
  
    /*  
     * Save the resulting new comp  
     */  
  
    save( NO_SHOW );  
    user_ack("Copy Complete", HELP_U_ACK);  
}  
  
return (OK);  
}
```

```
/******  
SWITCH_NAME
```

Purpose: Switch_name goes through the comp string and changes the old name to the new name. This routine puts the comp into a string until it finds the CompName, then it passes the comp name, and puts the rest of the Comp in another string. When finished doing this, it copies the first part of the comp into Comp, then cats the new comp name into Comp, then finally, cats the second part of the comp.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/8/88
1/20/89 rewritten, TAH

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

```
*****  
#define MAX_HEAD_LEN 300  
switch_name (oldName)  
{  
    char oldName[];  
  
    int i, /* Index into the Comp string */  
        l, /* Index into onePartOfComp */  
        r, /* Index into otherPartOfComp */  
        k, /* Index into oldName */  
        nameLength; /* The length of the comp name */  
  
    char onePartOfComp[MAX_HEAD_LEN], /* Comp string before oldName */  
        otherPartOfComp[MAX_COMP_LEN], /* Comp string after oldName */  
        foundName=FALSE; /* Flag */  
  
    /*  
     * Get the length of the comp name  
     */  
    nameLength = strlen (oldName);  
  
    /*  
     * Process through the Comp string looking  
     * for the old comp name.  
     */  
    i = k = l = r = 0;  
    while (Comp[i] != 0 && i < MAX_COMP_LEN)  
    {  
        /*  
         * If we have not yet found the name  
         * then continue to look for it.  
         */  
        if (!foundName)  
        {  
            /*  
             * If we don't find the comp name in  
             * MAX_HEAD_LEN chars then let's return  
             */  
            if (i >= MAX_HEAD_LEN)  
                return ERROR;  
  
            /*  
             * Add a char from the Comp string to  
             * a string called onePartOfComp.  
             */  
            onePartOfComp[l++] = Comp[i];  
  
            /*  
             * If we find a match, see if its the  
             * last char match we need to make.  
             */  
            if (oldName[k] == Comp[i++])  
            {
```

90/06/07
13:12:55

copy.c

5

```
/*
 * If we have matched nameLength chars then
 * we have found the old name.
 */
if (++k == nameLength)
{
    onePartOfComp[l-nameLength] = 0; /* terminate */
    foundName = TRUE;
}
else k = 0; /* reset the matched letter counter */
}
/*
 * Now that we have found the comp name let us
 * gather up the rest of the Comp string
 * into another string.
 */
else otherPartOfComp[r++] = Comp[i++];
}

otherPartOfComp[r] = 0; /* terminate */

/*
 * Copy the half preceding the old comp name into Comp, then cat the
 * the new comp name onto that, and add in the otherPart of the comp
 */
strcpy (Comp, onePartOfComp);
strcat (Comp, CompName);
strcat (Comp, otherPartOfComp);

return( OK );
}
```

90/06/07
13:12:57

cr_cascade.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/CascadeB.h>
#include "code.h"
#include "create.h"

/*****<----->*****/
*
* MODULE NAME:  cr_cascade( instance, parent, menu, label)
*
*
* MODULE FUNCTION:
*
*   Creates a Cascade widget and returns a pointer to the managed widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

Widget cr_cascade( instance, parent, menu, mnemonic, label )

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char    *instance,      /* The instance name of the widget. It uniquely
                        * defines the widget.
                        */
        *label,
        mnemonic;
Widget  menu,
        parent;         /* The parent widget to which the Cascade widget
                        * will be attached.
                        */

{
    static Arg    args[3];
    Widget  widget;

    XtSetArg( args[0], XmNsubMenuId, menu );
    XtSetArg( args[1], XmNmnemonic, mnemonic );
    XtSetArg( args[2], XmNfontList, Fnt_List_Btn );

    /*
     * Use the Xtoolkit intrinsic XtCreateManagedWidget to create the Cascade widget,
     * attach it to the parent, and initialize all arguments.
     */

    widget = XmCreateCascadeButton( parent, label, args, 3 );
    XtManageChild( widget );
    return ( widget );
}
```

90/06/07
13:12:59

cr_command.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/PushB.h>
#include "code.h"

/*****<---->*****/
*
* MODULE NAME:  cr_command( instance, parent, label, callback, id )
*
*
* MODULE FUNCTION:
*
*   Used to build command widgets which are placed within forms.  Returns a pointer
*   to the created widget.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

Widget cr_command (instance, parent, label, callback, id)

```
int      id;

char      *instance,
          *label;

Widget    parent;

XtCallbackList  callback;

/* This function returns the return value of the
 * XtCreateManagedWidget function call.  This will
 * be a pointer to a widget.
 */

/* The instance name of the widget.  It uniquely
 * defines the widget.
 */

/* The string which this command widget will display.
 * Note that if this pointer is NULL, the command
 * label will be a single blank.
 */

/* The parent widget to which the command widget will
 * be attached.
 */

/*
 * Callback function name.
 */
```

```
{
    Widget      widget;
    register int  n = 0;
    Arg         args[3];
    XmString     tcs;

    /*
     * Create compound string for the button text.
     */

    tcs = XmStringLtoRCreate( label,  XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType,  XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString, tcs ); n++;
    XtSetArg( args[n], XmNfontList,   Fnt_List_Btn ); n++;

    /*
     * Create the command button.
     */
```

90/06/07
13:12:59

cr_command.c

2

```
widget = (Widget) XmCreatePushButton( parent, instance, args, n );  
XtManageChild( widget );  
XmStringFree ( tcs );
```

```
/*  
 * Add the callback routine if specified.  
 */
```

```
if ( callback )  
    callback->closure = (caddr_t) id;  
XtAddCallbacks( widget, XmNactivateCallback, callback );
```

```
return( widget );  
}
```


90/06/07
13:13:01

cr_form.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/Form.h>
#include "create.h"

/*****<----->*****/
*
* MODULE NAME: cr_form( instance, parent, h_offset, v_offset)
*
*
* MODULE FUNCTION:
*
* Creates a form widget and returns a pointer to the managed widget.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

Widget cr_form( instance, parent, h_offset, v_offset )

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance; /* The instance name of the widget. It uniquely
 * defines the widget.
 */

Widget parent, /* The parent widget to which the form widget will
 * be attached.
 */

h_offset, /* If specified, the form widget will be offset
 * to the right of the specified widget. If NULL,
 * no offset will be in effect.
 */

v_offset; /* If specified, the form widget will be offset
 * below the specified widget. If NULL, no offset
 * will be in effect.
 */

{
/*
 * Define the array which will contain all arguments required to create the form
 * widget. This includes the horizontal and vertical offsets. Note that these
 * arguments are set to NULL because it is not known which will be used. Also note
 * that making this array static forces the following code to perform some seemingly
 * redundant assignments.
 */

static Arg args[5];
int count = 0;
Widget widget;

XtSetArg( args[0], XmNborderWidth, 0 ); count++;

if ( v_offset )
{
XtSetArg( args[count], XmNtopAttachment, XmATTACH_WIDGET ); count++;
XtSetArg( args[count], XmNtopWidget, v_offset ); count++;
}

if ( h_offset )
{
```

90/06/07
13:13:01

cr_form.c

2

```
XtSetArg( args[count], XmNleftAttachment, XmATTACH_WIDGET ); count++;  
XtSetArg( args[count], XmNleftWidget,      h_offset );      count++;  
}
```

```
/*  
 * Use the Xtoolkit intrinsic XtCreateManagedWidget to create the form widget,  
 * attach it to the parent, and initialize all arguments.  
 */
```

```
widget = XmCreateForm( parent, instance, args, count );  
XtManageChild( widget );  
return ( widget );  
}
```

90/06/07
13:13:03

cr_frame.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/Frame.h>
#include "create.h"

/*****<----->*****/
*
* MODULE NAME: cr_frame( instance, parent, h_offset, v_offset )
*
*
* MODULE FUNCTION:
*
* Creates a frame widget and returns a pointer to the managed widget.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

Widget cr_frame( instance, parent, h_offset, v_offset )

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance; /* The instance name of the widget. It uniquely
                 * defines the widget.
                 */

Widget parent, /* The parent widget to which the frame widget will
               * be attached.
               */
h_offset, /* If specified, the frame widget will be offset
           * to the right of the specified widget. If NULL,
           * no offset will be in effect.
           */
v_offset; /* If specified, the frame widget will be offset
           * below the specified widget. If NULL, no offset
           * will be in effect.
           */

{
/*
 * Define the array which will contain all arguments required to create the frame
 * widget. This includes the horizontal and vertical offsets. Note that these
 * arguments are set to NULL because it is not known which will be used. Also note
 * that making this array static forces the following code to perform some seemingly
 * redundant assignments.
 */

static Arg args[4];
int count = 0;
Widget widget;

if ( v_offset )
{
XtSetArg( args[count], XmNtopAttachment, XmATTACH_WIDGET ); count++;
XtSetArg( args[count], XmNtopWidget, v_offset ); count++;
}

if ( h_offset )
{
XtSetArg( args[count], XmNleftAttachment, XmATTACH_WIDGET ); count++;
XtSetArg( args[count], XmNleftWidget, h_offset ); count++;
}
```

90/06/07
13:13:03

cr_frame.c

2

```
/*  
 * Use the Xtoolkit intrinsic XtCreateManagedWidget to create the frame widget,  
 * attach it to the parent, and initialize all arguments.  
 */
```

```
widget = XmCreateFrame( parent, instance, args, count );  
XtManageChild( widget );  
return ( widget );  
}
```

90/06/07
13:13:05

cr_frm_cmd.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/PushButton>
```

```

/*****<----->*****/
*
* MODULE NAME:  cr_frm_cmd( instance, parent, label, h_offset, v_offset, callback, id )
*
*
* MODULE FUNCTION:
*
*   Used to build command widgets which are placed within forms.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

```
Widget cr_frm_cmd ( instance, parent, label, h_offset, v_offset, callback, id )

    int      id;

                                /* This function returns the return value of the
                                * XtCreateManagedWidget function call. This will
                                * be a pointer to a widget.
                                */

    char      *instance,
                                /* The instance name of the widget. It uniquely
                                * defines the widget.
                                */
                                *label;
                                /* The string which this command widget will display.
                                * Note that if this pointer is NULL, the command
                                * label will be a single blank.
                                */

    Widget    parent,
                                /* The parent widget to which the command widget will
                                * be attached.
                                */

                                h_offset,
                                v_offset;

    XtCallbackList  callback;
                                /*
                                * Callback function name.
                                */

{
    Widget      widget;
    register int  n = 0;
    Arg          args[6];
    XmString     tcs;

    /*
    * Create compound string for the button text.
    */

    tcs = XmStringLtoRCreate( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType, XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString, tcs ); n++;

    /*
    * Place the new widget relative to the horizontal and/or
    * vertical widgets.
    */

    if ( v_offset )
    {
        XtSetArg( args[n], XmNtopAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNtopWidget, v_offset ); n++;
    }

    if ( h_offset )
    {
        XtSetArg( args[n], XmNleftAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNleftWidget, h_offset ); n++;
    }

    /*
    * Create the command button.
    */

    widget = (Widget) XmCreatePushButton( parent, instance, args, n );
    XtManageChild( widget );
    XmStringFree ( tcs );

    /*
    * Add the callback routine if specified.
    */

    if ( callback )
        callback->closure = (caddr_t) id;
    XtAddCallbacks( widget, XmNactivateCallback, callback );

    return( widget );
}
```

90/06/07
13:13:07

cr_frm_rc.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/RowColumn.h>
#include "create.h"

/*****<----->*****/
*
* MODULE NAME: cr_frm_rc( instance, parent, columns, h_offset, v_offset )
*
*
* MODULE FUNCTION:
*
*   Creates a RowColumn widget and returns a pointer to the managed widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

Widget cr_frm_rc( instance, parent, columns, h_offset, v_offset )

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance; /* The instance name of the widget. It uniquely
                 * defines the widget.
                 */

Widget h_offset,
       v_offset,
       parent; /* The parent widget to which the RowColumn widget
               * will be attached.
               */

int columns;

{
    static Arg args[6];
    Widget widget;
    int n=2;

    XtSetArg( args[0], XmNpacking, XmPACK_COLUMN );
    XtSetArg( args[1], XmNnumColumns, columns );

    /*
     * Place the new widget relative to the horizontal and/or
     * vertical widgets.
     */

    if ( v_offset )
    {
        XtSetArg( args[n], XmNtopAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNtopWidget, v_offset ); n++;
    }

    if ( h_offset )
    {
        XtSetArg( args[n], XmNleftAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNleftWidget, h_offset ); n++;
    }

    /*
```

90/06/07
11:13:07

cr_frm_rc.c

2

```
* Use the Xtoolkit intrinsic XtCreateManagedWidget to create the RowColumn widget,  
* attach it to the parent, and initialize all arguments.  
*/
```

```
widget = XmCreateRowColumn( parent, instance, args, n );  
XtManageChild( widget );  
return ( widget );  
}
```


90/06/07
13:13:09

cr_frm_txt.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/Text.h>
#include "create.h"

/*****<----->*****/
*
* MODULE NAME: cr_frm_txt( instance, parent, text, h_offset, v_offset, scrolled, rows,
*   columns )
*
*
* MODULE FUNCTION:
*
*   Creates a text widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

Widget cr_frm_txt( instance, parent, text, h_offset, v_offset, scrolled, rows, columns )

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char   *instance, /* The instance name of the widget. It uniquely
                  * defines the widget.
                  */
        *text;

Widget h_offset,
        v_offset,
        parent; /* The parent widget to which the text widget will
                  * be attached.
                  */

int     columns,
        rows,
        scrolled;

(
Widget widget;
Arg     args[8];
int     i=4;

XtSetArg( args[0], XmNvalue,      text );
XtSetArg( args[1], XmNeditMode,  XmMULTI_LINE_EDIT );
XtSetArg( args[2], XmNrows,      rows );
XtSetArg( args[3], XmNcolumns,   columns );

/*
 * Place the new widget relative to the horizontal and/or
 * vertical widgets.
 */

if ( v_offset )
{
XtSetArg( args[i], XmNtopAttachment, XmATTACH_WIDGET ); i++;
XtSetArg( args[i], XmNtopWidget,    v_offset );         i++;
}

if ( h_offset )
{
```

90/06/07
13:13:09

cr_frm_txt.c

2

```
XtSetArg( args[i], XmNleftAttachment, XmATTACH_WIDGET ); i++;
XtSetArg( args[i], XmNleftWidget, h_offset ); i++;
}
```

```
/*
 * Use the Xtoolkit intrinsic XtCreateManagedWidget to create the text widget,
 * attach it to the parent, and initialize all arguments.
 */
```

```
if ( scrolled )
    widget = XmCreateScrolledText( parent, instance, args, i );
else
    widget = XmCreateText( parent, instance, args, i );
```

```
XtManageChild( widget );
return ( widget );
```

```
}
```

90/06/07
13:13:11

cr_label.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Label.h>
#include "code.h"
#include "create.h"

/*****<----->*****/
*
* MODULE NAME: cr_label( instance, parent, label, borderWidth, top, bottom, left, right )
*
*
* MODULE FUNCTION:
*
* Used to build label widgets which are placed within forms. Returns a pointer
* to the newly created and managed widget.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/
```

```
Widget cr_label ( instance, parent, label, borderWidth, top, bottom, left, right )

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance, /* The instance name of the widget. It uniquely
                 * defines the widget.
                 */
      *label;    /* The string which this label widget will display.
                 */

int    borderWidth; /*
 *
 */

Widget parent;      /* The parent widget to which the label widget will
 * be attached.
 */

int    top, bottom, left, right;

{
    Widget      widget;
    register int n = 0;
    Arg         args[12];
    XmString    tcs;

    /*
     * Create compound string for the label text.
     */

    tcs = XmStringLtoRCreate( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType, XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString, tcs ); n++;
    XtSetArg( args[n], XmNborderWidth, borderWidth ); n++;
    XtSetArg( args[n], XmNfontList, Fnt_List_Btn ); n++;

    /*
     * Place the new widget relatively within the form.
     */
}
```

90/06/07
13:13:11

cr_label.c

2

```
*/  
  
if ( top != IGNORE )  
{  
    XtSetArg( args[n], XmNtopAttachment, XmATTACH_POSITION ); n++;  
    XtSetArg( args[n], XmNtopPosition, top ); n++;  
}  
  
if ( bottom != IGNORE )  
{  
    XtSetArg( args[n], XmNbottomAttachment, XmATTACH_POSITION ); n++;  
    XtSetArg( args[n], XmNbottomPosition, bottom ); n++;  
}  
  
if ( left != IGNORE )  
{  
    XtSetArg( args[n], XmNleftAttachment, XmATTACH_POSITION ); n++;  
    XtSetArg( args[n], XmNleftPosition, left ); n++;  
}  
  
if ( right != IGNORE )  
{  
    XtSetArg( args[n], XmNrightAttachment, XmATTACH_POSITION ); n++;  
    XtSetArg( args[n], XmNrightPosition, right ); n++;  
}  
  
/*  
 * Create the label widget.  
 */  
  
widget = (Widget) XmCreateLabel( parent, instance, args, n );  
XtManageChild( widget );  
XmStringFree ( tcs );  
  
return( widget );  
}
```

90/06/07
13:13:13

cr_pixmap.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/Label.h>
#include "code_const.h"
```

```

/*****<----->*****/
*
* MODULE NAME: cr_pixmap( instance, parent, pixmap, top, bottom, left, right )
*
*
* MODULE FUNCTION:
*
* Used to build pixmap label widgets which are placed within forms. A pointer
* to the newly created and managed widget is returned.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/
```

```
Widget cr_pixmap( instance, parent, pixmap, top, bottom, left, right )
```

```
/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */
```

```
char *instance; /* The instance name of the widget. It uniquely
 * defines the widget.
 */
```

```
Widget parent; /* The parent widget to which the pixmap widget will
 * be attached.
 */
```

```
Pixmap *pixmap; /* The pixmap to display in this label widget.
 */
```

```
int top, bottom,
left, right;
```

```
{
Widget widget;
Arg args[10];
int n = 2;
```

```
XtSetArg( args[0], XmNlabelType, XmPIXMAP );
XtSetArg( args[1], XmNlabelPixmap, *pixmap );
```

```
/*
 * Place the new widget relatively within the form.
 */
```

```
if ( top != IGNORE )
{
XtSetArg( args[n], XmNtopAttachment, XmATTACH_POSITION ); n++;
XtSetArg( args[n], XmNtopPosition, top ); n++;
}
```

```
if ( bottom != IGNORE )
{
XtSetArg( args[n], XmNbottomAttachment, XmATTACH_POSITION ); n++;
XtSetArg( args[n], XmNbottomPosition, bottom ); n++;
}
```

90/06/07
13:13:13

cr_pixmap.c

2

```
if ( left != IGNORE )
{
    XtSetArg( args[n], XmNleftAttachment, XmATTACH_POSITION ); n++;
    XtSetArg( args[n], XmNleftPosition,   left ); n++;
}

if ( right != IGNORE )
{
    XtSetArg( args[n], XmNrightAttachment, XmATTACH_POSITION ); n++;
    XtSetArg( args[n], XmNrightPosition,   right ); n++;
}

/*
 * Create the label widget with a pixmap.
 */

widget = (Widget) XmCreateLabel( parent, instance, args, n );
XtManageChild( widget );

return( widget );
}
```

90/06/07
13:13:14

cr_popup.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/mwm.h>
#include <Xm/BulletinB.h>
```

```

/*****<----->*****/
*
* MODULE NAME:  cr_popup( instance, parent )
*
*
* MODULE FUNCTION:
*
*   Used to build bulletin board popup widgets.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

90/06/07
13:13:14

cr_popup.c

2

```
Widget cr_popup( instance, parent )
```

```
/* This function returns the return value of the  
 * XtCreateManagedWidget function call. This will  
 * be a pointer to a widget.  
 */
```

```
char    *instance;
```

```
/* The instance name of the widget. It uniquely  
 * defines the widget.  
 */
```

```
Widget  parent;
```

```
/* The parent widget to which the command widget will  
 * be attached.  
 */
```

```
{
```

```
Widget  widget;  
Arg      args[1];
```

```
/*  
 * Create the popup widget.  
 */
```

```
widget = XmCreateBulletinBoardDialog( parent, instance, NULL, 0 );
```

```
XtSetArg( args[0], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL );  
XtSetValues( XtParent( widget ), args, 1 );
```

```
return( widget );
```

```
}
```


90/06/07
13:13:16

cr_rel_cmd.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/PushB.h>
```

```

/*****<----->*****/
*
* MODULE NAME: cr_rel_cmd( instance, parent, label, callback, id, top, left )
*
*
* MODULE FUNCTION:
*
* Used to build command widgets which are placed relatively within forms. A
* pointer to the newly created and managed widget is returned.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/
```

Widget cr_rel_cmd (instance, parent, label, callback, id, top, left)

```
int id;
/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance,
/* The instance name of the widget. It uniquely
 * defines the widget.
 */
*label;
/* The string which this command widget will display.
 */

Widget parent;
/* The parent widget to which the command widget will
 * be attached.
 */

int left,
top;

XtCallbackList callback;
/*
 * Callback function name.
 */
```

```
{
Widget widget;
register int n = 0;
Arg args[6];
XmString tcs;
```

```
/*
 * Create compound string for the button text.
 */
```

```
tcs = XmStringLtoRCreate( label, XmSTRING_DEFAULT_CHARSET );
XtSetArg( args[n], XmNlabelType, XmSTRING ); n++;
XtSetArg( args[n], XmNlabelString, tcs ); n++;
```

```
/*
 * Place the new widget relatively within the form.
 */
```

```
XtSetArg( args[n], XmNtopAttachment, XmATTACH_POSITION ); n++;
```

90/06/07
13:13:16

cr_rel_cmd.c

2

```
XtSetArg( args[n], XmNtopPosition, top ); n++;
```

```
XtSetArg( args[n], XmNleftAttachment, XmATTACH_POSITION ); n++;
```

```
XtSetArg( args[n], XmNleftPosition, left ); n++;
```

```
/*
```

```
 * Create the command button.
```

```
*/
```

```
widget = (Widget) XmCreatePushButton( parent, instance, args, n );
```

```
XtManageChild( widget );
```

```
XmStringFree ( tcs );
```

```
/*
```

```
 * Add the callback routine if specified.
```

```
*/
```

```
if ( callback )
```

```
    callback->closure = (caddr_t) id;
```

```
XtAddCallbacks( widget, XmNactivateCallback, callback );
```

```
return( widget );
```

```
}
```

90/06/07
13:13:18

cr_rowcol.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/RowColumn.h>
#include "create.h"
```

```

/*****<----->*****/
*
* MODULE NAME: cr_rowcol( instance, parent, columns, orientation )
*
*
* MODULE FUNCTION:
*
* Creates a RowColumn widget.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/
```

```
Widget cr_rowcol( instance, parent, columns, orientation )
```

```

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance; /* The instance name of the widget. It uniquely
                 * defines the widget.
                 */

Widget parent; /* The parent widget to which the RowColumn widget
                * will be attached.
                */

int columns,
orientation;

{
    static Arg args[4];
    Widget widget;

    if ( orientation == XmHORIZONTAL )
        XtSetArg( args[0], XmNpacking, XmPACK_COLUMN );
    else
        XtSetArg( args[0], XmNpacking, XmPACK_COLUMN );

    XtSetArg( args[1], XmNnumColumns, columns );
    XtSetArg( args[2], XmNentryAlignment, XmALIGNMENT_CENTER );
    XtSetArg( args[3], XmNorientation, orientation );

    /*
     * Use the Xtoolkit intrinsic XtCreateManagedWidget to create the RowColumn widget,
     * attach it to the parent, and initialize all arguments.
     */

    widget = XmCreateRowColumn( parent, instance, args, 4 );
    XtManageChild( widget );
    return ( widget );
}
```

90/06/07
13:13:20

cr_separator.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/Separator.h>
#include "code_const.h"
```

```

/*****<----->*****/
*
* MODULE NAME: cr_separator( instance, parent, top, bottom, left, right )
*
*
* MODULE FUNCTION:
*
*   Used to build separator widgets.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/
```

Widget cr_separator(instance, parent, top, bottom, left, right)

```

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char    *instance;    /* The instance name of the widget. It uniquely
                        * defines the widget.
                        */

Widget  parent;        /* The parent widget to which the separator widget will
                        * be attached.
                        */

int top, bottom, left, right;

{
    Arg    args[8];
    int    n = 0;
    Widget widget;

    /*
     * Place the new widget relatively within the form.
     */

    if ( top != IGNORE )
    {
        XtSetArg( args[n], XmNtopAttachment, XmATTACH_POSITION ); n++;
        XtSetArg( args[n], XmNtopPosition,    top ); n++;
    }

    if ( bottom != IGNORE )
    {
        XtSetArg( args[n], XmNbottomAttachment, XmATTACH_POSITION ); n++;
        XtSetArg( args[n], XmNbottomPosition,    bottom ); n++;
    }

    if ( left != IGNORE )
    {
        XtSetArg( args[n], XmNleftAttachment, XmATTACH_POSITION ); n++;
        XtSetArg( args[n], XmNleftPosition,    left ); n++;
    }
}
```

90/06/07
13:13:20

cr_separator.c

2

```
if ( right != IGNORE )
{
    XtSetArg( args[n], XmNrightAttachment, XmATTACH_POSITION ); n++;
    XtSetArg( args[n], XmNrightPosition, right ); n++;
}

/*
 * Create the separator widget.
 */

widget = XmCreateSeparator( parent, instance, args, n );
XtManageChild( widget );

return( widget );
}
```

90/06/07
13:13:22

cr_text.c

1

```
#include <X11/Intrinsic.h>
#include <Xm/Text.h>
#include "create.h"
```

```

/*****<----->*****/
*
* MODULE NAME: cr_text( instance, parent, h_offset, v_offset, text, scrolled, rows, columns)
*
*
* MODULE FUNCTION:
*
* Creates a text widget.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/
```

```
Widget cr_text( instance, parent, h_offset, v_offset, text, scrolled, rows, columns )
```

```

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance, /* The instance name of the widget. It uniquely
                 * defines the widget.
                 */
      *text;

Widget parent, /* The parent widget to which the text widget will
               * be attached.
               */

      h_offset, /* If specified, the text widget will be offset
               * to the right of the specified widget. If NULL,
               * no offset will be in effect.
               */

      v_offset; /* If specified, the text widget will be offset
               * below the specified widget. If NULL, no offset
               * will be in effect.
               */

int columns,
  rows,
  scrolled;

{
  Widget widget;
  Arg args[8];
  int count = 4;

  XtSetArg( args[0], XmNvalue, text );
  XtSetArg( args[1], XmNeditMode, XmMULTI_LINE_EDIT );
  XtSetArg( args[2], XmNrows, rows );
  XtSetArg( args[3], XmNcolumns, columns );

  /*
   * Create a scrolled window with a text edit widget.
   */

  if ( scrolled )
```

90/06/07
13:13:22

cr_text.c

2

```
{
widget = XmCreateScrolledText( parent, instance, args, count );
XtManageChild( widget );

if ( v_offset )
{
XtSetArg( args[0], XmNtopAttachment, XmATTACH_WIDGET );
XtSetArg( args[1], XmNtopWidget, v_offset );
XtSetValues( XtParent(widget), args, 2 );
}
if ( h_offset )
{
XtSetArg( args[0], XmNleftAttachment, XmATTACH_WIDGET );
XtSetArg( args[1], XmNleftWidget, h_offset );
XtSetValues( XtParent(widget), args, 2 );
}
}

/*
 * Create a non-scrolled text edit widget.
 */
else
{
if ( v_offset )
{
XtSetArg( args[count], XmNtopAttachment, XmATTACH_WIDGET ); count++;
XtSetArg( args[count], XmNtopWidget, v_offset ); count++;
}

if ( h_offset )
{
XtSetArg( args[count], XmNleftAttachment, XmATTACH_WIDGET ); count++;
XtSetArg( args[count], XmNleftWidget, h_offset ); count++;
}

widget = XmCreateText( parent, instance, args, count );
XtManageChild( widget );
}

return ( widget );
}
```

90/06/07
13:13:24

cr_toggle.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/ToggleB.h>
#include "code.h"
#include "create.h"
```

```
/*-----*/
*
* MODULE NAME: cr_toggle( instance, parent, label, callback, arm, disarm )
*
* MODULE FUNCTION:
*   Creates a toggle button widget.
*
* SPECIFICATION DOCUMENTS:
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```


90/06/07
13:13:24

cr_toggle.c

2

```
Widget cr_toggle( instance, parent, label, callback, arm, disarm )

/* This function returns the return value of the
 * XtCreateManagedWidget function call. This will
 * be a pointer to a widget.
 */

char *instance, /* The instance name of the widget. It uniquely
                 * defines the widget.
                 */

      *label;

Widget parent; /* The parent widget to which the toggle widget
                * will be attached.
                */

int arm,
    disarm;

XtCallbackList callback; /*
 *
 */

{
    Widget widget;
    register int n = 0;
    Arg args[3];
    XmString tcs;

    /*
     * Create compound string for the button text.
     */

    tcs = XmStringLtoRCreate( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType, XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString, tcs ); n++;
    XtSetArg( args[n], XmNfontList, Fnt_List_Btn ); n++;

    widget = XmCreateToggleButton( parent, instance, args, n );
    XtManageChild( widget );
    XmStringFree ( tcs );

    /*
     * Add the callback routine if specified.
     */

    if ( callback )
    {
        callback->closure = (caddr_t) arm;
        XtAddCallbacks( widget, XmNarmCallback, callback );
        callback->closure = (caddr_t) disarm;
        XtAddCallbacks( widget, XmNdisarmCallback, callback );
    }

    return ( widget );
}
```

90/06/07
13:13:26

create.h

1

```
/*-----*/
*
* FILE NAME:    create.h
*
*
* FILE FUNCTION:
*
*   This file contains the function prototypes of the Motif "create" widget helper
*   routines.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   N/A
*
*
*-----*/
```

```
extern Widget  cr_cascade  (),
               cr_command  (),
               cr_form     (),
               cr_frame    (),
               cr_frm_cmd  (),
               cr_frm_rc   (),
               cr_frm_txt  (),
               cr_label    (),
               cr_pixmap   (),
               cr_popup    (),
               cr_rel_cmd  (),
               cr_rowcol   (),
               cr_separator(),
               cr_text     (),
               cr_toggle   ();
```

90/06/87
13:13:28

delete.c

1

/*****

DELETE

Purpose: Delete removes the last token in the Comp string.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/19/87

Version: 1.0

Project: CODE (Comp Development Environment)

External Interfaces

getTokenLen() -- returns the length of a given token
dcrVarList() -- decrements the occurrence count for a variable
delete() -- look up bub!
lookBackJack() -- check whether prev paren is preceded by function

*****/
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"

```
delete ()
{
    int      compLength,      /* length of the comp */
            tokenLength,     /* length of the token */
            i,                /* index into Comp */
            tokenNumber,      /* the number of the token to delete */
            j;                /* index into var_temp */

    char      var_temp[MAX_VAR_LEN]; /* the variable to delete */

    tokenNumber = PrevChoice [ChoiceCounter - 1];
    tokenLength = getTokenLen(tokenNumber);
    compLength = strlen (Comp);

    /*
     * If the token is a variable we need
     * to find and index to its first char.
     */

    if (tokenNumber == MSID || tokenNumber == SIGNAL || tokenNumber == LOCAL)
    {
        /*
         * Copy the variable into a temp string
         */

        for (j=0,i=compLength-tokenLength+1;i<compLength;i++)
            var_temp[j++] = Comp[i];

        var_temp[j] = 0; /* terminate string */

        /*
         * Decrement the occurrence of this variable
         * or delete if last one.
         */

        dcrVarList (var_temp);
    }

    /*
     * Shorten up the CODE comp string to its new more
     * diminutive stature.
     */

    Comp[compLength - tokenLength] = 0;
    ChoiceCounter--;
    NeedToSave = TRUE;

    /*
     * Reset some global flags & counters
     */

    if(tokenNumber == IF)
    {
        WhereAmI = CONSEQUENCE;
        Equation = RHS;
        NumberOfIfs--;
    }
    else if (tokenNumber == THEN)
    {
        WhereAmI = PREMISE;
        Equation = RHS;
    }
    else if (tokenNumber == AND || tokenNumber == OR ||
            tokenNumber == SET || tokenNumber == ELSE)
    {
        Equation = RHS;

        /*
         * Reset this NestedElseCheck so that ELSE is
         * enabled in next_inputs
         */

        if(tokenNumber == ELSE)
            NestedElseCheck[NumberOfIfs-NumberOfEndifs] = THEN;
    }
    else if(tokenNumber == END_IF)
```

90/06/07
13:13:28

delete.c

3

```
{
    NumberOfEndifs--;
    Equation = RHS;
}
else if (tokenNumber == R_PAREN)
{
    ParenCount++;

    /*
     * Call to lookBackJack() looks backwards to see
     * whether or not we are inside of a function.
     */

    if (FuncParenCount > 0 || lookBackJack ())
        FuncParenCount++;
}
else if (tokenNumber == L_PAREN)
{
    ParenCount--;

    /*
     * If we are working inside of a function expression
     * we need to maintain its paren count as well
     */

    if (FuncParenCount > 0)
        FuncParenCount--;

    /*
     * If the previous choice is a function then delete it
     * also since it is linked to this left paren.
     */

    if (PrevChoice[ChoiceCounter-1] == NOT      ||
        PrevChoice[ChoiceCounter-1] == COS      ||
        PrevChoice[ChoiceCounter-1] == ACOS     ||
        PrevChoice[ChoiceCounter-1] == SIN      ||
        PrevChoice[ChoiceCounter-1] == ASIN     ||
        PrevChoice[ChoiceCounter-1] == TAN      ||
        PrevChoice[ChoiceCounter-1] == ATAN     ||
        PrevChoice[ChoiceCounter-1] == POWER    ||
        PrevChoice[ChoiceCounter-1] == LOG      ||
        PrevChoice[ChoiceCounter-1] == EXP      ||
        PrevChoice[ChoiceCounter-1] == FUNCTION ||
        PrevChoice[ChoiceCounter-1] == Sqrt)

        delete ();
}
else if (tokenNumber == EQ ||
        tokenNumber == LT || tokenNumber == GT ||
        tokenNumber == LE || tokenNumber == GE ||
        tokenNumber == NE)
{
    Equation = LHS;
    NumberOfCompares--;
}
}
```

90/06/07
13:13:28

delete.c

4

```
/******
    lookBackJack
This is a short routine which looks back into the token history
to see if the paren we are deleting belongs to a function.
*****/

lookBackJack ()
{
    int    parenCount, /* Number of parens we've found so far */
           i;          /* index into PrevChoice */

    i = ChoiceCounter; /* start at the end */
    parenCount = 0;

    /*
     * Move backwards through the token string trying to
     * find the match to the paren we have in our hand.
     */

    do
    {
        if (PrevChoice[i] == R_PAREN)
        {
            parenCount++;
        }
        else if (PrevChoice[i] == L_PAREN)
        {
            parenCount--;
        }
    } while (parenCount > 0 && i-- > 0);

    /*
     * If the matching paren is just after a function,
     * then return TRUE.
     */

    if (PrevChoice[i-1] == NOT    || PrevChoice[i-1] == SQRT ||
        PrevChoice[i-1] == COS    || PrevChoice[i-1] == ACOS ||
        PrevChoice[i-1] == SIN    || PrevChoice[i-1] == ASIN ||
        PrevChoice[i-1] == TAN    || PrevChoice[i-1] == ATAN ||
        PrevChoice[i-1] == LOG    || PrevChoice[i-1] == EXP  ||
        PrevChoice[i-1] == POWER || PrevChoice[i-1] == FUNCTION)
        return (TRUE);
    else
        return (FALSE);
}
```

90/06/07
13:13:28

delete.c

5

```
/*  
    dcrVarList
```

Purpose: dcrVarList decrements the occurrence of a passed variable if it finds it in the CompVars structure. It will also remove the variable from the CompVars structure if the occurrence count is 0.

Returns: 0 if success, -1 if variable not found.

Designer: Troy Heindel/NASA

Programmer: Troy Heindel/NASA

Date: 11/20/88

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

```
*****  
dcrVarList (variableName)  
    char variableName[];  
{  
    int j, index;          /* loop counters, and indexing */  
  
    /*  
     * Get the index of the passed variable and decrement its  
     * occurrence counter by 1, and remove it if its the last  
     */  
    for (index=0; index<NumCompVars; index++)  
    {  
        if (strcmp (CompVars[index].name, variableName) == 0)  
        {  
            /*  
             * If there are now 0 occurrences of this variable we  
             * we remove it from the list of comp variables  
             */  
            if (--CompVars[index].occurrence == 0)  
            {  
                for (j=index; j<NumCompVars; j++)  
                {  
                    strcpy (CompVars[j].name, CompVars[j+1].name);  
                    strcpy (CompVars[j].type, CompVars[j+1].type);  
                    strcpy (CompVars[j].class, CompVars[j+1].class);  
                    CompVars[j].occurrence = CompVars[j+1].occurrence;  
                    CompVars[j].put_or_get = CompVars[j+1].put_or_get;  
                    CompVars[j].lo1_limit = CompVars[j+1].lo1_limit;  
                    CompVars[j].lo2_limit = CompVars[j+1].lo2_limit;  
                    CompVars[j].hi1_limit = CompVars[j+1].hi1_limit;  
                    CompVars[j].hi2_limit = CompVars[j+1].hi2_limit;  
                    strcpy(CompVars[j].nomenclature, CompVars[j+1].nomenclature);  
                }  
                --NumCompVars;  
            }  
            /*  
             * Let's break out since we found it  
             */  
            break;  
        }  
    }  
    return;  
}
```

90/06/07
13:13:28

delete.c

6

```
*****  
getTokenLen
```

Purpose: getTokenLen takes a token and figures out its string length, which is needed for adding and deleting tokens, and retrieving comps.

Designer: Troy Heindel/NASA

Programmer: Troy Heindel/NASA

Date: 5/29/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/  
getTokenLen (tokenClass)  
    int tokenClass; /* IF, THEN, ...etc... */  
{  
    int tokenLength,  
        compLength,  
        i;  
  
    /*  
     * Get the length of the entire comp string  
     */  
    tokenLength = 0;  
    compLength = strlen (Comp);  
  
    /*  
     * PRINT (case 1)  
     */  
    if(tokenClass == PRINT)  
    {  
        /*  
         * Find both sets of double quotes  
         */  
        for (i=0; i<2; i++)  
        {  
            /*  
             * Move backwards finding double quotes  
             */  
            do  
            {  
                tokenLength++;  
                compLength--;  
            } while (Comp[compLength] != '"' && compLength);  
        }  
  
        /*  
         * Add 8 for ' print# '  
         */  
        compLength -= 8;  
        tokenLength += 8;  
  
        /*  
         * Move backwards finding the previous token  
         */  
        while ((Comp[compLength-1] == ' ' || Comp[compLength-1] == '\n') && compLength)  
        {  
            tokenLength++;  
            compLength--;  
        }  
    }  
  
    /*  
     * COMMENT (case 2)  
     */  
    else if(tokenClass == COMMENT)
```


90/06/07
13:13:28

delete.c

7

```
(
/*
 * Find the start comment delimiter '/*'
 */
do
(
    tokenLength++;
    compLength--;
)while ((!(Comp[compLength] == '/' && Comp[compLength+1] == '*'))&&compLength);
)

/*
 * START & STOP
 */

else if ((tokenClass == START) || (tokenClass == STOP))
{
    /*
     * Find the left paren. Then subtract off "start("
     */

    while ( Comp[compLength] != '(' )
    {
        tokenLength++;
        compLength--;
    }

    if ( tokenClass == START )
        tokenLength += 6;
    else
        tokenLength += 5;

    while ((Comp[compLength] == ' ' ) || (Comp[compLength] == '\n'))
    {
        tokenLength++;
        compLength--;
    }
}

/*
 * STRING (case 4)
 */
else if(tokenClass == STRING)
{
    /*
     * Find both sets of double quotes
     */
    for (i=0;i<2;i++)
    (
        /*
         * Move backwards finding double quotes
         */
        do
        (
            tokenLength++;
            compLength--;
        ) while (Comp[compLength] != '"' && compLength);
    )

    /*
     * Add one for the space just before it
     */
    tokenLength++;
    compLength--;
}

/*
 * EVERYTHING ELSE
 * This else handles single word tokens.
 * i.e. not special case, e.g. 'if'
 */
else
(
    /*
     * Count up the chars in the char part
     */
    while (Comp[compLength] != ' ' && Comp[compLength] != '\n' && compLength)
    {
```

90/06/07
13:13:28

delete.c

8

```
        tokenLength++;  
        compLength--;  
    }  
  
    /*  
     * Count up the chars in the indent part  
     */  
    while (Comp[compLength-1] == ' ' || Comp[compLength-1] == '\\n' && compLength)  
    {  
        tokenLength++;  
        compLength--;  
    }  
    }  
    return (tokenLength);  
}
```

90/06/07
13:13:31

edit.c

1

```

/*****<----->*****/
*
* MODULE NAME:  edit( void )
*
*
* MODULE FUNCTION:
*
*   This function allows the user to edit the current group's 4 source files
*   that comprise the group.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
#include "widgets.h"

void edit()
{
    char temp[400],          /* Text message repository      */
        highLevelFile[PATH_LEN], /* Path for CODE lang. file    */
        compVarsFile [PATH_LEN], /* Path for comp variable file */
        compCFile    [PATH_LEN], /* Path for comp translation file */
        groupCFile   [PATH_LEN]; /* Path for the group source file */

    Dimension h, w, x, y;

    x = get_x      ( top );
    y = get_y      ( top );

    /*DEBUG
    w = get_width ( top );
    h = get_height( top );
    */

    /*
    * Create absolute paths to the high level,
    * variable, comp C, and group C files.
    */

    sprintf (highLevelFile, "%s/%s/%s.h", CodeGroups, GroupName, CompName);
    sprintf (compVarsFile,  "%s/%s/%s.v", CodeGroups, GroupName, CompName);
    sprintf (compCFile,     "%s/%s/%s.c", CodeGroups, GroupName, CompName);
    sprintf (groupCFile,    "%s/%s/%s.c", CodeGroups, GroupName, GroupName);

    /*
    * Make the system call which calls the editor, checking, of
    * of course, for errors.
    */

    if ( SetNum == 1 )
    {
        sprintf( temp, "xterm -geometry %dx%d+%d+%d ", 155, 43, x, y );
        strcat ( temp, "-bg lightblue -fn fixed -e vi " );
    }
    else
    {
        sprintf( temp, "xterm -geometry %dx%d+%d+%d ", 124, 40, x, y );
    }
}
```

90/06/07
13:13:31

edit.c

2

```
    strcat ( temp, "-bg lightblue -fn 9x15 -e vi " );  
}  
  
strcat ( temp, highLevelFile ); strcat( temp, " " );  
strcat ( temp, compVarsFile ); strcat( temp, " " );  
strcat ( temp, compCFile ); strcat( temp, " " );  
strcat ( temp, groupCFile ); strcat( temp, " " );  
strcat ( temp, "2>>/tmp/code.err" );  
  
if ( system(temp) == ERROR )  
    user_ack( "Unable to complete system call" );  
}
```

90/06/07
13:13:33

get_header.c

1

GET_HEADER

Purpose: Get_header gets the header info on a new comp.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/17/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

External Interfaces

cleanSlate() -- Initializes global variables.
displayWA() -- displays a string in the work area window

*****/

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <pwd.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
```

*****<----->*****

* MODULE NAME: get_header ()

* MODULE FUNCTION:

* Get_header gets the header info on a new comp.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

```
get_header ()
{
    time_t timesec;           /* seconds since Jan 1, 1970 */
    long int days,hrs,mins,secs; /* used in time calculations */
    int     getuid (),        /* UNIX routine to get user information */
           rc;
    char    time_str[50];     /* String to build time with */
    struct passwd *getpwuid (); /* UNIX routine to get user information */
    struct passwd *user_pass; /* A structure to put the user info into */

    /*
     * Initialize variables.
     */

    cleanSlate();
```

90/06/07
13:13:33

get_header.c

2

```
/*
 * Append the current time to the Comp string.
 */

strcpy (Comp, "/*****\n");

timesec = time( (time_t *) 0 );
days   = timesec/86400;
hrs     = timesec/3600;
mins    = timesec/60;
secs    = timesec - mins*60;
mins    = mins - hrs*60;
hrs     = hrs - days*24;
hrs     = hrs - 5;

sprintf(time_str, "%03d:%02d:%02d", days, hrs, mins, secs);
strcat (Comp, " Creation Time: ");
strcat (Comp, time_str);

/*
 * Append the user's name.
 */

strcat (Comp, " Author: ");
user_pass = getpwuid (getuid());
strcat (Comp, user_pass->pw_gecos);
strcat (Comp, "\n");

/*
 * Append the group & comp names.
 */

strcat (Comp, " Group Name: ");
strcat (Comp, GroupName);
strcat (Comp, "\t\t Comp Name: ");
strcat (Comp, CompName);
strcat (Comp, "\n");

/*
 * Append the purpose to the Comp string within
 * the global CompInfo structure.
 */

strcat (Comp, " Purpose: ");
displayWA(Comp);

rc = ABORT;
do {
    rc = get_string("What is the purpose of this computation?", CompInfo[CompNumber].purpose, 80, TRUE);
    if ( rc == ABORT )
        user_ack("You really should (must) add a purpose ", HELP_U_ACK);
} while ( rc == ABORT );

strcat (Comp, CompInfo[CompNumber].purpose);
strcat (Comp, "\n*****/");

/*
 * Set the PrevChoice to COMMENT since the header is just
 * a large comment.
 */
PrevChoice[ChoiceCounter++] = COMMENT;
NeedToSave = TRUE;
Disposition = INCOMPLETE;
}
```

90/06/07
13:13:35

get_name.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/MessageB.h>
#include "code.h"
#include "widgets.h"
#include "create.h"

extern XtCallbackRec clear_get_name[];
extern char Group_Str[],
            Comp_Str[];

/*****<----->*****/
*
* MODULE NAME: get_name( numOfItems, theType, nameToGet, numToGet, disposition )
*
*
* MODULE FUNCTION:
*
* Function allows the user to select either an existing group or comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/
int get_name( numOfItems, theType, nameToGet, numToGet, disposition )

int *disposition,
    numOfItems,
    *numToGet;
char *nameToGet,
    *theType;

{
    char button_string[60],
        message[100];
    int i;
    Arg args[7];

    struct group_info_struct nameList[ MAX_COMPS ]; /* Place to put the names */

    /*
     * Let's leave if there is nothing to list.
     */

    if ( numOfItems < 1 )
    {
        sprintf( message, "There are not any %s(s) to list", theType );
        user_ack( message, HELP_U_ACK );
        return( ERROR );
    }

    /*
     * Need to get a Comp name.
     */

    if ( strcmp(theType, "Comp") == 0 )
    {
        /*
         * If there is only one Comp, then it is the one, return its name.
         */
    }
}
```

90/06/07
13:13:15

get_name.c

2

```
if ( numOfItems < 2 )
{
    strcpy( nameToGet, CompInfo[0].name );
    *disposition = CompInfo[0].disposition;
    *numToGet = 0;
    return( OK );
}

/*
 * Otherwise, there is more than one Comp to chose from, build a command
 * button for each Comp.
 */

select_cursor( Clock_Cursor );
for ( i=0; i<numOfItems; i++ )
{
    strcpy( nameList[i].name, CompInfo[i].name );
    nameList[i].disposition = CompInfo[i].disposition;
}

/*
 * Set the popup label to Comp_Str.
 */

XtSetArg(args[0],XmNlabelString,XmStringLtoRCreate(Comp_Str,XmSTRING_DEFAULT_CHARSET));
XtSetValues( lbl_gname, args, 1 );
PopupHelp = HELP_C_SEL;
}

/*
 * Need to get a Group name.
 */

else if ( strcmp(theType,"Group") == 0 )
{
    /*
     * If there is only one Group, then it is the one, return its name.
     */

    if ( numOfItems < 2 )
    {
        strcpy( nameToGet, GroupInfo[0].name );
        *disposition = GroupInfo[0].disposition;
        *numToGet = 0;
        readCompNames();
        return( OK );
    }

    /*
     * Otherwise, there is more than one Group to chose from, build a command
     * button for each Group.
     */

    select_cursor( Clock_Cursor );
    for ( i=0; i<numOfItems; i++ )
        nameList[i] = GroupInfo[i];

    /*
     * Set the popup label to Group_Str.
     */

    XtSetArg(args[0],XmNlabelString,XmStringLtoRCreate(Group_Str,XmSTRING_DEFAULT_CHARSET));
    XtSetValues( lbl_gname, args, 1 );
    PopupHelp = HELP_G_SEL;
}

/*
 * Build the a set of command buttons and insert them into the 'get_name' window.
 */

i = 0;
while ((i<numOfItems) && (i<MAX_NAMES))
{
    sprintf( button_string, "%-16s", nameList[i].name );
    strcat( button_string, " " );
    switch( nameList[i].disposition )
```


90/06/07
13:13:35

3

get_name.c

```
{
case INCOMPLETE:   strcat( button_string, " INCOMPLETE " );
                   break;
case COMPLETE:     strcat( button_string, " COMPLETE " );
                   break;
case ERROR:        strcat( button_string, " ERROR " );
                   break;
case INSTALLED:    strcat( button_string, " INSTALLED " );
                   break;
default:           printf( "disposition: %d \n",nameList[i].disposition );
}

/*
 * Put all of the selection buttons in the first column, current release of
 * HP widgets will not work correctly with multiple columns.
 * TJB 07/07/89
 */

gname[i] = cr_command( NULLS, rcl_gname, button_string, clear_get_name, i );
i++;

if ((i<numOfItems) && (i==MAX_NAMES))
{
    select_cursor( Shuttle_Cursor );
    user_ack( "ERROR: Cannot display all available group names.", HELP_U_ACK );
    for (i=0; i<MAX_NAMES; i++)
        XtUnmanageChild( gname[i] );
    return( ERROR );
}

/*
 * Display the 'get_name' popup. After the user has made a selection, kill all the
 * widgets which were used to construct the Comp/Group buttons.
 */

select_cursor( Shuttle_Cursor );
process_popup( dlg_gname, WAIT );

select_cursor( Clock_Cursor );
for (i=0; i<numOfItems; i++)
    XtUnmanageChild( gname[i] );
select_cursor( Shuttle_Cursor );

/*
 * If the user aborted the Comp/Group selection screen, return an error,
 * otherwise, return the number of the selected Comp/Group, copy the corresponding
 * name of the Comp/Group, and if the user selected a Group, then read in the
 * Comps associated with the Group.
 */

if ( Get_Name_Stat == ABORT )
    return( ABORT );
else
{
    *numToGet    = Get_Name_Stat;
    *disposition = nameList[Get_Name_Stat].disposition;
    strcpy( nameToGet, nameList[Get_Name_Stat].name );
    if ( strcmp( theType, "Group" ) == 0 )
        readCompNames();
    return( OK );
}
}
```

90/06/07
13:13:37

graphics.c

1

```
*****<----->*****
*
* FILE NAME:    graphics.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which maintain the X Windows user interface.  This
*   file does not contain all the graphics routines, see the following files for
*   additional X type stuff:
*
*       init_gp.c
*       utilities.c
*       window_io.c
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   CreateDefaultImage - creates an image from a bit array
*   arm_toggle()        - sets the toggle state to armed
*   disarm_toggle()     - sets the toggle state to disarmed
*   get_height()        - retrieves the height of the specified widget (pixels)
*   get_width()         - retrieves the width  of the specified widget (pixels)
*   get_x()             - retrieves the x coordinate for the specified widget
*   get_y()             - retrieves the y coordinate for the specified widget
*   popup_wait()        - waits for user to select a button in the popup
*   process_popup()     - handles display of popups
*   set_cycle_mode()    - sets the cycle selection mode indicator of the current comp
*
*****<----->*****/
```

```
#include <stdio.h>
#include "X11/Intrinsic.h"
#include <Xm/Xm.h>
#include "code.h"
#include "widgets.h"
```

```
Arg args[1];
```

90/06/07
13:13:37

graphics.c

2

```

/*****<----->*****/
*
* MODULE NAME:  arm_toggle( widget )
*
* MODULE FUNCTION:
*
*   Sets the specified toggle state to armed.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void arm_toggle( widget )
{
    XmToggleButtonSetState( widget, TRUE, FALSE );
}
```

```

/*****<----->*****/
*
* MODULE NAME: CreateDefaultImage( bits, width, height )
*
*
* MODULE FUNCTION:
*
*   Create an image from a bit array.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   OSF MOTIF widget source code - version 1.0
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

```
XImage *CreateDefaultImage (bits, width, height)

    char *bits;
    int   width, height;

{
    XImage *image;

    image = (XImage *) XtMalloc (sizeof (XImage));
    image->width      = width;
    image->height      = height;
    image->data        = bits;
    image->depth        = 1;
    image->xoffset      = 0;
    image->format       = XYBitmap;
    image->byte_order   = LSBFirst;
    image->bitmap_unit  = 8;
    image->bitmap_pad   = 8;
    image->bytes_per_line = (width+7)/8;
    image->bitmap_bit_order = LSBFirst;

    return (image);
}
```

90/06/07
13:13:37

graphics.c

4

```

/*****<----->*****/
*
* MODULE NAME:  disarm_toggle( widget )
*
*
* MODULE FUNCTION:
*
*   Sets the specified toggle's state to disarmed.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void disarm_toggle( widget )
{
    XmToggleButtonSetState( widget, FALSE, FALSE );
}
```

90/06/07
13:13:37

graphics.c

5

```

/*****<----->*****/
*
* MODULE NAME:  get_height( widget )
*
*
* MODULE FUNCTION:
*
*   Retrieves the height (in pixels)  of the specified widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

Dimension get_height( widget )

    Widget widget;

{
    Arg args[1];

    XtSetArg ( args[0], XmNheight, (XtArgVal) NULL );
    XtGetValues( widget, args, 1 );
    return( (Dimension) args[0].value );
}
```

90/06/07
13:13:37

graphics.c

6

```

/*****<----->*****/
*
* MODULE NAME:  get_width( widget )
*
*
* MODULE FUNCTION:
*
*   Function retrieves the width of the specified widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

Dimension get_width( widget )
{
    Widget widget;

    (
        XtSetArg    ( args[0], XmNwidth, (XtArgVal) NULL );
        XtGetValues( widget, args, 1 );
        return( (Dimension) args[0].value );
    )
}
```

90/06/07
13:13:37

graphics.c

7

```

/*****<----->*****/
*
* MODULE NAME:  get_x( widget )
*
* MODULE FUNCTION:
*
*   Retrieves the x coordinate for the specified widget.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

Dimension get_x( widget )

    Widget widget;

(
    XtSetArg    ( args[0], XmNx, (XtArgVal) NULL );
    XtGetValues( widget, args, 1 );
    return( (Dimension) args[0].value );
)
```


90/06/07
13:13:37

graphics.c

8

```

/*****<----->*****/
*
* MODULE NAME:  get_y( widget )
*
*
* MODULE FUNCTION:
*
*   Retrieves the y coordinate for the specified widget.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

Dimension get_y( widget )

    Widget widget;

{
    XtSetArg ( args[0], XmNy, (XtArgVal) NULL );
    XtGetValues( widget, args, 1 );
    return( (Dimension) args[0].value );
}
```

90/06/07
13:13:37

graphics.c

9

```
/*-----*/
*
* MODULE NAME:  popup_wait()
*
* MODULE FUNCTION:
*
*   Function processes events locally (as opposed to XtMainLoop) until the user
*   selects a button in the currently active popup.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
void popup_wait()
{
    XEvent event;

    ActivePopup++;
    PopupStat[ ActivePopup ] = -1;

    while ( PopupStat[ ActivePopup ] < 0 )
    {
        XtNextEvent( &event );
        XtDispatchEvent( &event );
    }

    ActivePopup--;
}
```

90/06/07
13:13:37

graphics.c

10

```

/*****<----->*****/
*
* MODULE NAME:  process_popup( widget )
*
*
* MODULE FUNCTION:
*
*   Function places the pointer in the middle of the Comp Builder screen, then
*   places the popup and if necessary, waits for the user's response.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

```

```
void process_popup( widget, wait_flag )
```

```
Widget widget;
```

```
{
```

```
Dimension x,y;
```

```
x = get_x( form ) + ( get_width(form)/2);
```

```
y = get_y( form ) + (get_height(form)/2);
```

```
XWarpPointer( XtDisplay(top), None, XtWindow(form), 0, 0, 0, 0, x, y );
XtManageChild( widget );
```

```
if ( wait_flag )
    popup_wait();
```

```
}
```

90/06/07
13:13:37

graphics.c

11

```
/*-----*/
*
* MODULE NAME:  set_cycle_mode( mode )
*
*
* MODULE FUNCTION:
*
*   Routine sets the cycle selection mode indicator of the current comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*-----*/

void set_cycle_mode( mode )

    int mode;

{
    if ( Disposition == NO_GROUP )
    {
        user_ack( "No group selected, use CREATE to start a new Group/Comp", HELP_U_ACK );
        disarm_toggle( tgl_cycle );
        disarm_toggle( tgl_lshot );
        return;
    }
    else
    {
        CompInfo[CompNumber].cycle_mode = mode;
        NeedToSave = TRUE;
    }
}
```

90/06/07
13:13:39

hisde.h

1

```
/*-----*/
*
* FILE NAME:    hisde.h
*
*
* FILE FUNCTION:
*
*   This file contains the constants used to send messages to the h_message()
*   client of the SwRI Hardware Independent Software Development Environment (HISDE).
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* FILE MODULES:
*
*   N/A
*
/*-----*/

/*
 * Constants for use by various message types.
 */

#define MSG_APPLICATION      1
#define MSG_ERROR            2
#define MSG_HOST             3
#define MSG_INFORMATION      4
#define MSG_WARNING          5

/*
 * Maximum message length.
 */

#define MAX_MESSAGE_LENGTH 160
```

90/06/07
13:13:41

init_code.c

1

```
*****<----->*****
*
* FILE NAME:    init_code.c
*
*
* FILE FUNCTION:
*
*   Initializes the paths and files required by CODE.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   CheckOrMkdir() - makes sure the directory exists, if not, creates it
*   init_code()   - initializes paths to files used by CODE
*
*****<----->*****/
```

```
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
#include "hilde.h"
```

90/06/07
13:13:41

init_code.c

2

```

/*****<----->*****/
*
* MODULE NAME:  CheckOrMkdir( path )
*
*
* MODULE FUNCTION:
*
*   Function checks for a directory, if it exists, the routines exits.  If the
*   directory does not exist, this function tries to create it.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   12/11/86
*
/*****<----->*****/

int CheckOrMkdir( path )

    char *path;

    (
        char message[200];

        if ( access(path,F_OK) == ERROR )
            if ( mkdir(path,511) != OK )
                {
                    sprintf( message, "CODE: could not create directory: %s", path );
                    user_ack( message, HELP_U_ACK );
                    return( ERROR );
                }

        return( OK );
    )

```

90/06/07
13:13:41

init_code.c

3

```

/*****<----->*****/
*
* MODULE NAME:  init_code()
*
*
* MODULE FUNCTION:
*
*   Initializes paths to the various files required by CODE.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

int  init_code()
{
    int      rc = OK;
    FILE     *ptr;
    char     ch1,
            ch2,
            listUserFuncs[ 250 ],
            message[ 200 ];

    /*
     * Check for the CODE error file, if present, delete it.
     */

    if ( access("/tmp/code.err",F_OK) == OK )
        if ( system("rm /tmp/code.err >>/tmp/code.err 2>&1") != OK )
            user_ack( "CODE: unable to remove /tmp/code.err", HELP_U_ACK );

    /*
     * Get the terminal setup.  Save terminal setup parms so they can be restored when
     * CODE is exited.
     */

    /*
     * Load the knowledge base for CLIPS.
     */

    /*
     * Get environment variable which specifies path to CODE root directory.  Build paths
     * to support directories.
     */

    if ( getenv("RTDS") == NULL )
    {
        user_ack( "CODE: RTDS environment variable not defined", HELP_U_ACK );
        if ( get_string("Enter path to RTDS",RTDS,79,FALSE) == ABORT )
            return( ERROR );
    }
    else
        strcpy( RTDS, getenv("RTDS") );

    sprintf (AM,          "%s/rtds/am",          RTDS);          /* directory */
    sprintf (AMSupport,    "%s/rtds/am/support",  RTDS);          /* directory */
}
```


4

```
MSIDCount++;
```

```
    }
    fclose (ptr);
}

/*
 * Create and execute the system command which
 * will create a file listing of user functions.
 */

sprintf (listUserFuncs, "ar t %s > /tmp/user_funcs 2>/tmp/code.err", UserFuncsLib);
system(listUserFuncs); /* If we can't do this, who cares! */

/*
 * Now create a path for the file to be read
 */

strcpy (listUserFuncs, "/tmp/user_funcs");

/*
 * Open the newly created file /tmp/user_funcs
 * and read out the names of the user functions.
 *
 * Start at 0.
 */

NumberOfUserFuncs = 0;
if (ptr = fopen(listUserFuncs, "r", "init_code"))
{
    /*
     * Loop through the file, filling the global structure MSIDTable
     * with all those lovely msid's and related info.
     */

    fscanf (ptr, "%*[^\\n]"); /* Rip the header */
    while (fscanf (ptr, "%s", UserFuncs[NumberOfUserFuncs]) != EOF)
    {
        UserFuncs[NumberOfUserFuncs][strlen(UserFuncs[NumberOfUserFuncs])-2]='\\0';

        /*
         * Make sure the function is not a CODE reserved word.
         */

        if (strcmp ("if",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("then",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("and",     UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("or",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("not",     UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print1",  UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print2",  UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print3",  UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print4",  UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("print5",  UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("set",     UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp (">",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp (">=",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("<",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("<=",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("=",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("else",   UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("endif",  UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("",       UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("(",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp (")",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("+",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("-",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("**",     UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("/",      UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("bitXor", UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("exp",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("log",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("cos",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("acos",   UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("sin",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("asin",   UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("tan",    UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("atan",   UserFuncs[NumberOfUserFuncs]) == 0 ||
            strcmp ("sqrt",   UserFuncs[NumberOfUserFuncs]) == 0 ||
```

init_code.c

```
strcmp ("power", UserFuncs[NumberOfUserFuncs]) == 0 ||
strcmp ("bitAnd", UserFuncs[NumberOfUserFuncs]) == 0 ||
strcmp ("bitOr", UserFuncs[NumberOfUserFuncs]) == 0 ||
strcmp ("shiftL", UserFuncs[NumberOfUserFuncs]) == 0 ||
strcmp ("shiftR", UserFuncs[NumberOfUserFuncs]) == 0 ||
strcmp ("p", UserFuncs[NumberOfUserFuncs]) == 0 ||
strcmp ("", UserFuncs[NumberOfUserFuncs]) == 0)

{
    strcpy( message, "CODE: " );
    strcat( message, UserFuncs[NumberOfUserFuncs] );
    strcat( message, " is a CODE reserved word and therefore can not be used as a" );
    strcat( message, " user function." );

#ifdef HISDE
    h_message( MSG_WARNING, message );
#endif

    user_ack( message, HELP_U_ACK );
    sprintf( message, "CODE: Ignoring: %s", UserFuncs[NumberOfUserFuncs] );

#ifdef HISDE
    h_message( MSG_WARNING, message );
#endif
}

/*
 * Increment the function counter.
 */
else NumberOfUserFuncs++;
}

/*
 * Let's not forget to clean up after ourselves.
 */

if(system ("rm /tmp/user_funcs 2>>/tmp/code.err")!=OK)
    user_ack( "CODE: unable to remove /tmp/user_funcs", HELP_U_ACK );

/*
 * Read the GroupNames file into memory. If we have a problem
 * we exit since no useful work can be done without reading
 * and writing this file.
 */

if (readGroupNames() == ERROR)
    cleanExit();

/*
 * Initialize some variables to a known state.
 */

Comp[0]          = NULL;
CompNumber       = ERROR;
Disposition      = NO_GROUP;
GroupNumber      = ERROR;
NeedToSave       = FALSE;
NumOfComps       = 0;
theMode          = RUNmode;
WaitCursor       = 0;

strcpy( CompName, " " );
strcpy( GroupName, " " );

/*
 * Update the status window.
 */

put_status();

return( OK );
}
```

90/06/07
13:13:55

next_inputs.c

1

NEXT_INPUTS

Purpose: Next_Inputs is used to determine what options are valid after each choice.

Designer: J. Harold Taylor/SDC

Programmer: J. Harold Taylor/SDC

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

External Interfaces

*****/

/*****

Include files

*****/

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
```

next_inputs (on_my_left,on_my_right)

int on_my_left,on_my_right;

```
{
    int    i,
           defaults = 0;
```

```
    if (on_my_right == IGNORE)
        return (OK);
```

```
    TotValPts = defaults;
```

/*****

COMMENTS

Comments are handled differently than other tokens, since they do not change what is valid, but are part of the comp string. Search backward for a valid token, or just find the end of the string.

*****/

```
if (on_my_left == COMMENT)
```

```
{
```

```
    /*****
    Start out assuming its a new comp
    *****/
```

```
    on_my_left = CREATE;
```

```
    /*****
    Loop through the choices until we find one which
    is not a comment, then reset and break
    *****/
```

```
    for (i=ChoiceCounter-1;i>1;i--)
```

```
    {
```

```
        /*****
        Once we have found the first occurrence of
        something that is not a comment we call, break
        *****/
```

```
        if (PrevChoice[i] != COMMENT)
```

```
        {
```

```
            on_my_left = PrevChoice[i];
            break;
```

```
        }
```

```
    }
```

```
}
```

/*****

IF

*****/

```
if (on_my_left == IF)
{
    TotValPts = defaults + 20;
    ValidPoints[defaults + 0] = LOCAL;
    ValidPoints[defaults + 1] = MSID;
    ValidPoints[defaults + 2] = SIGNAL;
    ValidPoints[defaults + 3] = NUMBER;
    ValidPoints[defaults + 4] = L_PAREN;
    ValidPoints[defaults + 5] = NOT;
    ValidPoints[defaults + 6] = COS;
    ValidPoints[defaults + 7] = ACOS;
    ValidPoints[defaults + 8] = SIN;
    ValidPoints[defaults + 9] = ASIN;
    ValidPoints[defaults + 10] = TAN;
    ValidPoints[defaults + 11] = ATAN;
    ValidPoints[defaults + 12] = STRING;
    ValidPoints[defaults + 13] = PI;
    ValidPoints[defaults + 14] = FUNCTION;
    ValidPoints[defaults + 15] = SQRT;
    ValidPoints[defaults + 16] = POWER;
    ValidPoints[defaults + 17] = EXP;
    ValidPoints[defaults + 18] = LOG;
    ValidPoints[defaults + 19] = SUBTRACT;
    LikelyNextChoice = MSID;
}
/*****
MSID NUMBER PI CHAR SIGNAL LOCAL
*****/
else if (on_my_left == MSID || on_my_left == NUMBER ||
on_my_left == PI || on_my_left == STRING ||
on_my_left == SIGNAL || on_my_left == LOCAL)
{
    /*****
    Having an equal number of ifs and endifs means
    we're outside of the if which is the same
    as being in the consequence.
    *****/
    if (WhereAmI == CONSEQUENCE || NumberOfIfs == NumberOfEndifs)
    {
        if (Equation == RHS)
        {
            if (CompareType[NumberOfCompares-1][0] != 'c')
            {
                /*****
                Check if they have balanced their paren's,
                if not, make 'em balance the parens.
                This takes care of argDefs too.
                *****/
                if (ParenCount > 0)
                {
                    TotValPts = defaults + 10;
                    ValidPoints[defaults + 0] = ADD;
                    ValidPoints[defaults + 1] = SUBTRACT;
                    ValidPoints[defaults + 2] = MULTIPLY;
                    ValidPoints[defaults + 3] = DIVIDE;
                    ValidPoints[defaults + 4] = SHIFTR;
                    ValidPoints[defaults + 5] = SHIFTL;
                    ValidPoints[defaults + 6] = R_PAREN;
                    ValidPoints[defaults + 7] = BITOR;
                    ValidPoints[defaults + 8] = BITXOR;
                    ValidPoints[defaults + 9] = BITAND;
                    LikelyNextChoice = R_PAREN;
                    if (FuncParenCount > 0 &&
                        FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
                        ValidPoints[TotValPts++] = COMMA;
                }
            }
            else
            {
                TotValPts = defaults + 14;
                ValidPoints[defaults + 0] = ADD;
                ValidPoints[defaults + 1] = SUBTRACT;
                ValidPoints[defaults + 2] = MULTIPLY;
                ValidPoints[defaults + 3] = DIVIDE;
                ValidPoints[defaults + 4] = IF;
                ValidPoints[defaults + 5] = SET;
                ValidPoints[defaults + 6] = PRINT;
                ValidPoints[defaults + 7] = SHIFTR;
                ValidPoints[defaults + 8] = SHIFTL;
```

next_inputs.c

```

ValidPoints[defaults + 9] = BITOR;
ValidPoints[defaults + 10] = BITXOR;
ValidPoints[defaults + 11] = BITAND;
ValidPoints[defaults + 12] = START;
ValidPoints[defaults + 13] = STOP;
LikelyNextChoice = IF;
/*****
If an ELSE has already been used in this if
we don't let them use it again!
*****/
if(NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
{
    LikelyNextChoice = ELSE;
    ValidPoints[TotValPts++] = ELSE;
}
/*****
If they have closed all their if's with
endif's then they don't need endif any more.
*****/
if ((NumberOfIfs - NumberOfEndifs) > 0)
{
    LikelyNextChoice = END_IF;
    ValidPoints[TotValPts++] = END_IF;
}
}
/*
* Compare type is a string.
*/
else
{
    TotValPts = defaults + 5;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SET;
    ValidPoints[defaults + 2] = PRINT;
    ValidPoints[defaults + 3] = START;
    ValidPoints[defaults + 4] = STOP;
    /*****
    If an ELSE has already been used in this if
    we don't let them use it again!
    *****/
    if(NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    {
        LikelyNextChoice = ELSE;
        ValidPoints[TotValPts++] = ELSE;
    }
    /*****
    If they have closed all their if's with
    endif's then they don't need endif any more.
    *****/
    if ((NumberOfIfs - NumberOfEndifs) > 0)
    {
        LikelyNextChoice = END_IF;
        ValidPoints[TotValPts++] = END_IF;
    }
}
}
/*****
Don't worry - be happy --- we couldn't get to
this point with an MSID or constant expression.
See on_my_left == SET.
*****/
else if (Equation == LHS)
{
    TotValPts = defaults + 1;
    ValidPoints[defaults + 0] = EQ;
    LikelyNextChoice = EQ;
}
}
else if (WhereAmI == PREMISE)
{
    if (Equation == LHS)
    {
        if (ParenCount > 0)
        {
            /*****
            If we're done defining arguments, let
            them do some relational stuff.

```

next_inputs.c

```
*****
if (FuncParenCount == 0)
{
    TotValPts = defaults + 16;
    ValidPoints[defaults + 0] = ADD;
    ValidPoints[defaults + 1] = SUBTRACT;
    ValidPoints[defaults + 2] = MULTIPLY;
    ValidPoints[defaults + 3] = DIVIDE;
    ValidPoints[defaults + 4] = LT;
    ValidPoints[defaults + 5] = GT;
    ValidPoints[defaults + 6] = LE;
    ValidPoints[defaults + 7] = GE;
    ValidPoints[defaults + 8] = NE;
    ValidPoints[defaults + 9] = R_PAREN;
    ValidPoints[defaults + 10] = EQ;
    ValidPoints[defaults + 11] = SHIFTR;
    ValidPoints[defaults + 12] = SHIFTL;
    ValidPoints[defaults + 13] = BITOR;
    ValidPoints[defaults + 14] = BITXOR;
    ValidPoints[defaults + 15] = BITAND;

    LikelyNextChoice = EQ;
}
else
{
    TotValPts = defaults + 10;
    ValidPoints[defaults + 0] = ADD;
    ValidPoints[defaults + 1] = SUBTRACT;
    ValidPoints[defaults + 2] = MULTIPLY;
    ValidPoints[defaults + 3] = DIVIDE;
    ValidPoints[defaults + 4] = SHIFTR;
    ValidPoints[defaults + 5] = SHIFTL;
    ValidPoints[defaults + 6] = R_PAREN;
    ValidPoints[defaults + 7] = BITOR;
    ValidPoints[defaults + 8] = BITXOR;
    ValidPoints[defaults + 9] = BITAND;
    if (FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
        ValidPoints[TotValPts++] = COMMA;
    LikelyNextChoice = R_PAREN;
}
}
else
{
    TotValPts = defaults + 15;
    ValidPoints[defaults + 0] = ADD;
    ValidPoints[defaults + 1] = SUBTRACT;
    ValidPoints[defaults + 2] = MULTIPLY;
    ValidPoints[defaults + 3] = DIVIDE;
    ValidPoints[defaults + 4] = LT;
    ValidPoints[defaults + 5] = GT;
    ValidPoints[defaults + 6] = LE;
    ValidPoints[defaults + 7] = GE;
    ValidPoints[defaults + 8] = NE;
    ValidPoints[defaults + 9] = EQ;
    ValidPoints[defaults + 10] = SHIFTR;
    ValidPoints[defaults + 11] = SHIFTL;
    ValidPoints[defaults + 12] = BITOR;
    ValidPoints[defaults + 13] = BITXOR;
    ValidPoints[defaults + 14] = BITAND;
    LikelyNextChoice = EQ;
}
}
else if (Equation == RHS)
{
    if (ParenCount > 0)
    {
        /*****
        If we're done defining function arguments,
        let them do some relational stuff.
        *****/
        if (FuncParenCount == 0)
        {
            TotValPts = defaults + 15;
            ValidPoints[defaults + 0] = AND;
            ValidPoints[defaults + 1] = DIVIDE;
            ValidPoints[defaults + 2] = OR;
            ValidPoints[defaults + 3] = ADD;
            ValidPoints[defaults + 4] = SUBTRACT;
```

next_inputs.c

```

ValidPoints[defaults + 5] = MULTIPLY;
ValidPoints[defaults + 6] = BITXOR;
ValidPoints[defaults + 7] = R_PAREN;
ValidPoints[defaults + 8] = BITAND;
ValidPoints[defaults + 9] = BITOR;
ValidPoints[defaults + 10] = SHIFTR;
ValidPoints[defaults + 11] = SHIFTL;
LikelyNextChoice = R_PAREN;
}
else
{
    TotValPts = defaults + 10;
    ValidPoints[defaults + 0] = DIVIDE;
    ValidPoints[defaults + 1] = ADD;
    ValidPoints[defaults + 2] = SUBTRACT;
    ValidPoints[defaults + 3] = MULTIPLY;
    ValidPoints[defaults + 4] = R_PAREN;
    ValidPoints[defaults + 5] = SHIFTR;
    ValidPoints[defaults + 6] = SHIFTL;
    ValidPoints[defaults + 7] = BITOR;
    ValidPoints[defaults + 8] = BITXOR;
    ValidPoints[defaults + 9] = BITAND;

    if (FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
        ValidPoints[TotValPts++] = COMMA;
    LikelyNextChoice = R_PAREN;
}
}
else
{
    TotValPts = defaults + 12;
    ValidPoints[defaults + 0] = AND;
    ValidPoints[defaults + 1] = DIVIDE;
    ValidPoints[defaults + 2] = OR;
    ValidPoints[defaults + 3] = ADD;
    ValidPoints[defaults + 4] = SUBTRACT;
    ValidPoints[defaults + 5] = MULTIPLY;
    ValidPoints[defaults + 6] = BITXOR;
    ValidPoints[defaults + 7] = THEN;
    ValidPoints[defaults + 8] = BITAND;
    ValidPoints[defaults + 9] = BITOR;
    ValidPoints[defaults + 10] = SHIFTR;
    ValidPoints[defaults + 11] = SHIFTL;
    LikelyNextChoice = THEN;
}
}
}
}
}
/*****
Logical ops covers <, >, <=, >=, =, <>
*****/
else if ((on_my_left == GT) || (on_my_left == LT) ||
        (on_my_left == GE) || (on_my_left == LE) ||
        (on_my_left == NE) || (on_my_left == EQ) ||
        (on_my_left == ADD) || (on_my_left == SUBTRACT) ||
        (on_my_left == MULTIPLY) || (on_my_left == DIVIDE) ||
        (on_my_left == SHIFTL) || (on_my_left == SHIFTR) ||
        (on_my_left == BITOR) || (on_my_left == BITAND))
{
    TotValPts = defaults + 8;
    ValidPoints[defaults + 0] = MSID;
    ValidPoints[defaults + 1] = SIGNAL;
    ValidPoints[defaults + 2] = LOCAL;
    ValidPoints[defaults + 3] = FUNCTION;
    ValidPoints[defaults + 4] = NUMBER;
    ValidPoints[defaults + 5] = L_PAREN;
    ValidPoints[defaults + 6] = SUBTRACT;
    ValidPoints[defaults + 7] = STRING;

    if ((WhereAmI == CONSEQUENCE && CompareType[NumberOfCompares-1][0] == 'd') ||
        WhereAmI == PREMISE)
    {
        ValidPoints[TotValPts++] = COS;
        ValidPoints[TotValPts++] = ACOS;
        ValidPoints[TotValPts++] = SIN;
        ValidPoints[TotValPts++] = ASIN;
        ValidPoints[TotValPts++] = TAN;
        ValidPoints[TotValPts++] = ATAN;
    }
}

```



```
ValidPoints[TotValPts++] = PI;
ValidPoints[TotValPts++] = SQRT;
ValidPoints[TotValPts++] = POWER;
ValidPoints[TotValPts++] = EXP;
ValidPoints[TotValPts++] = LOG;
}
LikelyNextChoice = NUMBER;
)
/*****
AND, OR, BITXOR, BITAND, BITOR
*****/
else if (on_my_left == OR || on_my_left == BITXOR || on_my_left == AND )
(
    TotValPts = defaults + 20;
    ValidPoints[defaults + 0] = SIGNAL;
    ValidPoints[defaults + 1] = LOCAL;
    ValidPoints[defaults + 2] = MSID;
    ValidPoints[defaults + 3] = L_PAREN;
    ValidPoints[defaults + 4] = NOT;
    ValidPoints[defaults + 5] = NUMBER;
    ValidPoints[defaults + 6] = STRING;
    ValidPoints[defaults + 7] = PI;
    ValidPoints[defaults + 8] = COS;
    ValidPoints[defaults + 9] = ACOS;
    ValidPoints[defaults + 10] = SIN;
    ValidPoints[defaults + 11] = ASIN;
    ValidPoints[defaults + 12] = TAN;
    ValidPoints[defaults + 13] = ATAN;
    ValidPoints[defaults + 14] = FUNCTION;
    ValidPoints[defaults + 15] = SQRT;
    ValidPoints[defaults + 16] = POWER;
    ValidPoints[defaults + 17] = EXP;
    ValidPoints[defaults + 18] = LOG;
    ValidPoints[defaults + 19] = SUBTRACT;
    LikelyNextChoice = MSID;
)
/*****
THEN
*****/
else if (on_my_left == THEN)
(
    TotValPts = defaults + 5;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = PRINT;
    ValidPoints[defaults + 2] = SET;
    ValidPoints[defaults + 3] = START;
    ValidPoints[defaults + 4] = STOP;
    LikelyNextChoice = PRINT;
)
/*****
ELSE
*****/
else if (on_my_left == ELSE)
(
    TotValPts = defaults + 8;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SIGNAL;
    ValidPoints[defaults + 2] = LOCAL;
    ValidPoints[defaults + 3] = MSID;
    ValidPoints[defaults + 4] = PRINT;
    ValidPoints[defaults + 5] = SET;
    ValidPoints[defaults + 6] = START;
    ValidPoints[defaults + 7] = STOP;
    LikelyNextChoice = PRINT;
)
/*****
NOT
*****/
else if (on_my_left == NOT)
(
    TotValPts = defaults + 4;
    ValidPoints[defaults + 0] = MSID;
    ValidPoints[defaults + 1] = SIGNAL;
    ValidPoints[defaults + 2] = LOCAL;
    ValidPoints[defaults + 3] = L_PAREN;
    LikelyNextChoice = MSID;
)
/*****/
```

```
        L_PAREN
        *****/
    else if (on_my_left == L_PAREN)
    {
        TotValPts = defaults + 20;
        ValidPoints[defaults + 0] = MSID;
        ValidPoints[defaults + 1] = SIGNAL;
        ValidPoints[defaults + 2] = LOCAL;
        ValidPoints[defaults + 3] = NUMBER;
        ValidPoints[defaults + 4] = STRING;
        ValidPoints[defaults + 5] = NOT;
        ValidPoints[defaults + 6] = PI;
        ValidPoints[defaults + 7] = COS;
        ValidPoints[defaults + 8] = ACOS;
        ValidPoints[defaults + 9] = SIN;
        ValidPoints[defaults + 10] = ASIN;
        ValidPoints[defaults + 11] = TAN;
        ValidPoints[defaults + 12] = ATAN;
        ValidPoints[defaults + 13] = FUNCTION;
        ValidPoints[defaults + 14] = SQRT;
        ValidPoints[defaults + 15] = POWER;
        ValidPoints[defaults + 16] = EXP;
        ValidPoints[defaults + 17] = LOG;
        ValidPoints[defaults + 18] = L_PAREN;
        ValidPoints[defaults + 19] = SUBTRACT;
        LikelyNextChoice = MSID;
    }
    /*****
        COMMA
        *****/
    else if (on_my_left == COMMA)
    {
        TotValPts = defaults + 20;
        ValidPoints[defaults + 0] = MSID;
        ValidPoints[defaults + 1] = SIGNAL;
        ValidPoints[defaults + 2] = LOCAL;
        ValidPoints[defaults + 3] = NUMBER;
        ValidPoints[defaults + 4] = STRING;
        ValidPoints[defaults + 5] = NOT;
        ValidPoints[defaults + 6] = PI;
        ValidPoints[defaults + 7] = COS;
        ValidPoints[defaults + 8] = ACOS;
        ValidPoints[defaults + 9] = SIN;
        ValidPoints[defaults + 10] = ASIN;
        ValidPoints[defaults + 11] = TAN;
        ValidPoints[defaults + 12] = ATAN;
        ValidPoints[defaults + 13] = FUNCTION;
        ValidPoints[defaults + 14] = SQRT;
        ValidPoints[defaults + 15] = POWER;
        ValidPoints[defaults + 16] = EXP;
        ValidPoints[defaults + 17] = LOG;
        ValidPoints[defaults + 18] = L_PAREN;
        ValidPoints[defaults + 19] = SUBTRACT;
        LikelyNextChoice = MSID;
    }
    /*****
        R_PAREN
        *****/
    else if (on_my_left == R_PAREN)
    {
        /*****
         / + - * SHIFTL SHIFTR BITOR BITXOR BITAND
         are always allowed after a R_PAREN.
         *****/
        TotValPts = defaults + 10;
        ValidPoints[defaults + 0] = DIVIDE;
        ValidPoints[defaults + 1] = ADD;
        ValidPoints[defaults + 2] = SUBTRACT;
        ValidPoints[defaults + 3] = MULTIPLY;
        ValidPoints[defaults + 4] = SHIFTR;
        ValidPoints[defaults + 5] = SHIFTL;
        ValidPoints[defaults + 6] = BITOR;
        ValidPoints[defaults + 7] = BITXOR;
        ValidPoints[defaults + 8] = BITAND;

        if (FuncParenCount == 0)
        {
            /*****

```

90/06/07
13:13:55

next_inputs.c

8

```
If we're outside of an "if" statement
and all paren's are balanced and on
the RHS allow IF PRINT and SET.
*****/
if (NumberOfIfs == NumberOfEndifs && ParenCount == 0 && Equation == RHS)
{
    TotValPts = defaults + 15;
    ValidPoints[defaults + 10] = IF;
    ValidPoints[defaults + 11] = PRINT;
    ValidPoints[defaults + 12] = SET;
    ValidPoints[defaults + 13] = START;
    ValidPoints[defaults + 14] = STOP;
    LikelyNextChoice = IF;
}
/*****
If we're in the premise and not defining
function arguments allow AND OR BITXOR
BITAND BITOR.
*****/
else if (WhereAmI == PREMISE)
{
    /*****
    If we're in the PREMISE and our overall paren
    count is balanced and we're on the RHS allow
    some logical operands .
    *****/
    if (ParenCount == 0)
    {
        if (Equation == RHS)
        {
            TotValPts = defaults + 13;
            ValidPoints[defaults + 10] = AND;
            ValidPoints[defaults + 11] = OR;
            ValidPoints[defaults + 12] = THEN;
            LikelyNextChoice = THEN;
        }
        /*****
        If we also happen to be on the LHS of an Equation
        allow some relational stuff too.
        *****/
        else if (Equation == LHS)
        {
            TotValPts = defaults + 16;
            ValidPoints[defaults + 10] = LT;
            ValidPoints[defaults + 11] = GT;
            ValidPoints[defaults + 12] = LE;
            ValidPoints[defaults + 13] = GE;
            ValidPoints[defaults + 14] = NE;
            ValidPoints[defaults + 15] = EQ;
            LikelyNextChoice = EQ;
        }
    }
    /*****
    If we're in the PREMISE and our overall paren
    count is unbalanced allow R_PAREN and logical
    operands.
    *****/
    else
    {
        TotValPts = defaults + 13;
        ValidPoints[defaults + 10] = AND;
        ValidPoints[defaults + 11] = OR;
        ValidPoints[defaults + 12] = R_PAREN;
        LikelyNextChoice = R_PAREN;
    }
}
/*****
We're in the CONSEQUENCE and not defining function
arguments.
*****/
else
{
    /*****
    If we're in the CONSEQUENCE and not defining
    function arguments and we have balanced paren's
    allow ENDIF PRINT SET.
    *****/
    if (ParenCount == 0)
```

```
{
    TotValPts = defaults + 16;
    ValidPoints[defaults + 10] = END_IF;
    ValidPoints[defaults + 11] = PRINT;
    ValidPoints[defaults + 12] = SET;
    ValidPoints[defaults + 13] = IF;
    ValidPoints[defaults + 14] = START;
    ValidPoints[defaults + 15] = STOP;
    LikelyNextChoice = END_IF;
    /*****
    If we don't already have an else associated with
    the current THEN allow an ELSE.
    *****/
    if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    {
        ValidPoints[TotValPts++] = ELSE;
        LikelyNextChoice = ELSE;
    }
}
else
{
    /*****
    If we don't have balanced paren's be better give
    them the opportunity to balance them.
    *****/
    TotValPts = defaults + 11;
    ValidPoints[defaults + 10] = R_PAREN;
    LikelyNextChoice = R_PAREN;
}
}
/*****
We're defining function arguments.
*****/
else
{
    TotValPts = defaults + 11;
    ValidPoints[defaults + 10] = R_PAREN;
    if (FunctionArgsDef[FunctionCurrent] < FunctionArguments[FunctionCurrent]-1)
        ValidPoints[TotValPts++] = COMMA;
    LikelyNextChoice = R_PAREN;
}
}
/*****
SET
*****/
else if (on_my_left == SET)
{
    TotValPts = defaults + 2;
    ValidPoints[defaults + 0] = SIGNAL;
    ValidPoints[defaults + 1] = LOCAL;
    LikelyNextChoice = SIGNAL;
}

/*****
*   START & STOP
*****/
else if ((on_my_left == START) || (on_my_left == STOP))
{
    TotValPts = defaults;
    ValidPoints[TotValPts++] = IF;
    ValidPoints[TotValPts++] = SET;
    ValidPoints[TotValPts++] = PRINT;
    ValidPoints[TotValPts++] = START;
    ValidPoints[TotValPts++] = STOP;
    LikelyNextChoice = IF;

    if ((NumberOfIfs - NumberOfEndifs) > 0)
    {
        ValidPoints[TotValPts++] = END_IF;
        LikelyNextChoice = END_IF;

        /*****
        *   If we don't already have an else associated with
        *   the current THEN allow an ELSE.
        *****/
    }
}
```

```
if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
{
    ValidPoints[TotValPts++] = ELSE;
    LikelyNextChoice = ELSE;
}
}

/*****
PRINT
*****/
else if (on_my_left == PRINT)
{
    /*****
    If an ELSE has already been used in this if
    we don't let them use it again!
    *****/
    if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
        TotValPts = defaults + 6;
    else TotValPts = defaults + 6;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SET;
    ValidPoints[defaults + 2] = PRINT;
    ValidPoints[defaults + 3] = START;
    ValidPoints[defaults + 4] = STOP;
    LikelyNextChoice = PRINT;
    if (NumberOfIfs - NumberOfEndifs > 0)
    {
        TotValPts += 1;
        ValidPoints[TotValPts - 1] = END_IF;
        LikelyNextChoice = END_IF;
        /*****
        If we don't already have an else associated with
        the current THEN allow an ELSE.
        *****/
        if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
        {
            TotValPts += 1;
            ValidPoints[TotValPts - 1] = ELSE;
            LikelyNextChoice = ELSE;
            LikelyNextChoice = ELSE;
        }
    }
}

/*****
CREATE, DOCUMENT
*****/
else if (on_my_left == CREATE || on_my_left == HELP)
{
    TotValPts = defaults + 5;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SET;
    ValidPoints[defaults + 2] = PRINT;
    ValidPoints[defaults + 3] = START;
    ValidPoints[defaults + 4] = STOP;
    LikelyNextChoice = IF;
}

/*****
END_IF
*****/
else if (on_my_left == END_IF)
{
    TotValPts = defaults + 5;
    ValidPoints[defaults + 0] = IF;
    ValidPoints[defaults + 1] = SET;
    ValidPoints[defaults + 2] = PRINT;
    ValidPoints[defaults + 3] = START;
    ValidPoints[defaults + 4] = STOP;
    /*****
    If the difference between the if and endifs
    is 0, then we don't need any more endifs.
    If an ELSE has already been used in this if
    we don't let them use it again!
    *****/
    if ((NumberOfIfs-NumberOfEndifs) > 0)
    {
        ValidPoints[TotValPts++] = END_IF;
        LikelyNextChoice = IF;
    }
}
```

90/06/07
13:13:55

next_inputs.c

11

```
    }
    if (NestedElseCheck[NumberOfIfs-NumberOfEndifs] == THEN)
    {
        ValidPoints[TotValPts++] = ELSE;
        LikelyNextChoice = ELSE;
    }
}
/*****
    DELETE
*****/
if (on_my_right == DELETE)
    LikelyNextChoice = DELETE;
/*****
    For when we just have started up.
*****/
if (on_my_right == 99)
    TotValPts = 0;
}
```

1

90/06/07
13:13:58

2

pixmap.h

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

[illegible]

90/06/07
13:14:00

put_local.c

1

PUT_LOCAL

Purpose: Put_local handles the local variables.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/27/87

Version: 2.0

Project: CODE (Comp Development Environment)

Revised by: Troy A. Heindel -- 10-27-88

Reasons for Revision:

External Interfaces

*****/

Include files

*****/

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
```

```

put_local ()
{
    int      localIndex,      /* The index to the variable in the CompVars struct */
            globalIndex;     /* This index is not of use to local variables */

    char      local[MAX_VAR_LEN], /* String for temporary variable storage */
            msgStr[150];

    /*
     * Get the name the local from the user
     */

    if (getVarName (local, LOCAL) == ERROR)
        return (ERROR);

    /*
     * Now check to see if the name is proper
     */

    if (var_check (local, &localIndex, &globalIndex, "local") == ERROR)
    {
        sprintf(msgStr, "%s has been used previously as a Signal variable", local);
        strcat (msgStr, "Signal and Local variables can not share names. ");
        user_ack (msgStr, HELP_U_ACK);
        return ERROR;
    }

    /*
     * Check to see that the type comparison is correct.
     * If we are on the right hand side of an Equation,
     * the local is defined, and the types aren't kosher.
     */

    if (Equation == RHS && localIndex != -1 &&
        type_check(CompVars[localIndex].type[0]) == ERROR)
    {
        sprintf (msgStr, "%s is of wrong type. Should have been of type '%s'", local, CompareType[NumberOfCompar
es-1]);
        user_ack (msgStr, HELP_U_ACK);
        return ERROR;
    }

    /*
     * Clean up indentation and add msid to comp
     */

    indent (PREMISE);
    strcat (Comp, local);
    updateWA (local);

    /*
     * Place the token choice into the history of
     * token choices in case of delete.
     */

    PrevChoice[ChoiceCounter++] = LOCAL;

    /*
     * If the local variable was not previously used
     * in this comp it will have a local index = -1.
     * If this is the case we need to get type info.
     */

    if (localIndex != -1)
    {
        ++CompVars[localIndex].occurrence;
        /*
         * if (LHS) then set the CompareType to type of this local
         */
        if (Equation == LHS)
            strcpy (CompareType[NumberOfCompares++], CompVars[localIndex].type);
    }
    else /* localIndex == -1 */
    {
        CompVars[NumCompVars].lo1_limit = 0;
        CompVars[NumCompVars].lo2_limit = 25;
        CompVars[NumCompVars].hi1_limit = 50;
    }
}

```

90/06/07
13:14:00

put_local.c

3

```
CompVars[NumCompVars].hi2_limit = 100;
CompVars[NumCompVars].put_or_get = GET;
strcpy (CompVars[NumCompVars].name, local);
strcpy (CompVars[NumCompVars].class, "local");
CompVars[NumCompVars].occurrence = 1;
sprintf(CompVars[NumCompVars].nomenclature, "/* %s */", CompVars[NumCompVars].name);
/*****
  If LHS then we get the type set the CompareType to it
  and increment the NumberOfCompares counter
  *****/
if (Equation == LHS)
{
    get_type(CompVars[NumCompVars].type, "Of what type?");
    strcpy (CompareType[NumberOfCompares++], CompVars[NumCompVars].type);
}
/*****
  If RHS then we default the local to the CompareType
  *****/
else
{
    strcpy (CompVars[NumCompVars].type, CompareType[NumberOfCompares-1]);
}
NumCompVars++; /* increment the count of comp variables */
} /* end of new local definition */
NeedToSave = TRUE;
}
```

90/06/07
13:14:00

put_local.c

4

```
getVarName(nameToGet, class)
char *nameToGet;
int class; /* Either LOCAL, MSID, or SIGNAL */

{
    int rc;

    /*****
    Keep getting name until we get a valid one
    *****/
    do
    {
        if ((rc = get_string("Enter the variable name", nameToGet, MAX_VAR_LEN-1, FALSE)) != ABORT)
        {
            upper( nameToGet );
            /*
            * Now that we have a variable, make sure it doesn't
            * contain unwanted characters.
            */
            rc = goodVar(nameToGet);
        }

        /*
        * First check to see if they just want to leave
        */

        if ( rc == ABORT )
            return ERROR;

    } while (rc == ERROR);

    return OK;
}
```

90/06/07
13:14:00

put_local.c

5

```
goodVar(varName)
char varName[];
{
    int i; /* Simple Counter */

    /*
     * Never allow the first character to be
     * a number.
     */

    if(varName[0] >= '0' && varName[0] <= '9')
    {
        user_ack("Your input string contained an illegal character", HELP_U_ACK);
        return ERROR;
    }

    /*
     * Does the string contain unwanted chars?
     * Allow only alphaNumerics and underscore.
     */

    for (i=0;i<strlen(varName);i++)
    {
        if (((varName[i] >= 'A' && varName[i] <= 'Z') ||
            (varName[i] >= 'a' && varName[i] <= 'z') ||
            (varName[i] >= '0' && varName[i] <= '9') ||
            varName[i] == '_'))
        {
            user_ack("Your input string contained an illegal character", HELP_U_ACK);
            return ERROR;
        }
    }
    return OK;
}
```

90/06/07
13:14:02

put_msid.c

1

PUT_MSID

Purpose: Put_msid validates and adds the entered
to the comp string and resets all the
neccessary flags.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 6/20/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/
/*****
```

Include files

```
*****/
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
```

put_msid ()

```
{
    int local_index, /* The index to the variable in the CompVars struct */
        global_index; /* The index to the variable in the MSIDTable */

    char msid[MSID_NAME_LEN], /* String used to store the name of the msid */
        type[TYPE_LEN], /* A place to put type of the msid */
        nomenclature[NOMEN_LEN], /* A place to put the msid's nomenclature */
        reply[50],
        message[150]; /* String used for displaying message to the user */

    /*
     * Get the name the msid from the user
     */
    if (getVarName (msid, MSID) == ERROR)
        return ERROR;

    /*
     * Try and get an index for the msid
     */
    var_check (msid, &local_index, &global_index, "msid");

    /*****
     * If it's defined locally, show them the nomenclature and
     * increment the occurence count.
     *****/
    if (local_index != ERROR)
    {
        /*
         * Show 'em the nomenclature for a second.
         */

        sprintf ( message, "%s", CompVars[local_index].nomenclature );
        user_ack( message, HELP_U_ACK );

        /*****
         * Set up type in case we're on the LHS so we can set the
         * compare type.
         *****/
        strcpy(type, CompVars[local_index].type);
        if (Equation == RHS && type_check(type[0]) == ERROR)
        {
            sprintf (message, "%s is of wrong type. Should have been of type '%s' ", msid, CompareType[Numbe
rofCompares-1]);
        }
    }
}
```

90/06/07
13:14:02

put_msid.c

2

```
user_ack(message, HELP_U_ACK);
return ERROR;
}
else
    CompVars[local_index].occurrence++;
}
/*****
If it's defined globally and not defined locally show them the
nomenclature define it locally using the global information.
*****/
else if (global_index != ERROR && local_index == ERROR)
{
    /*
    * Show 'em the nomenclature for a second.
    */

    sprintf ( message, "%s", MSIDTable[global_index].nomenclature );
    user_ack( message, HELP_U_ACK );

    strcpy(type, MSIDTable[global_index].type);
    /*****
    Check to see that the type comparison is correct
    *****/
    if (Equation == RHS && type_check(type[0]) == ERROR)
    {
        sprintf (message, "%s is of wrong type. Should have been of type '%s'", msid, CompareType[Number
OfCompares-1]);
        user_ack(message, HELP_U_ACK);
        return ERROR;
    }
    else
    {
        strcpy(CompVars[NumCompVars].name, msid);
        strcpy(CompVars[NumCompVars].type, type);
        strcpy(CompVars[NumCompVars].nomenclature, MSIDTable[global_index].nomenclature);
        strcpy(CompVars[NumCompVars].class, "msid");
        CompVars[NumCompVars].occurrence = 1;
        CompVars[NumCompVars].put_or_get = GET;
        CompVars[NumCompVars].lo1_limit = 0;
        CompVars[NumCompVars].lo2_limit = 25;
        CompVars[NumCompVars].hi1_limit = 50;
        CompVars[NumCompVars].hi2_limit = 100;
        NumCompVars++;
    }
}
/*****
If it's not defined globally - and not defined locally warn them
and get the type and nomenclature info and define it locally.
*****/
else if(global_index == ERROR && local_index == ERROR)
{
    sprintf(message,"Warning: %s is not a defined telemetry point on this system",msid);
    user_ack(message, HELP_U_ACK);
    get_type(type,"Of what type?");
    /*****
    Check to see that the type comparison is correct
    *****/
    if (Equation == RHS && type_check(type[0]) == ERROR)
    {
        sprintf (message,"%s is of wrong type. Should have been of type '%s' ", msid, CompareType[Number
OfCompares-1]);
        user_ack(message, HELP_U_ACK);
        return ERROR;
    }
    else
    {
        strcpy(CompVars[NumCompVars].name, msid);
        strcpy(CompVars[NumCompVars].type, type);
        get_string("Please enter msid nomenclature",reply,49,FALSE);
        sprintf(nomenclature, " /* %s */", reply);
        strcpy(CompVars[NumCompVars].nomenclature, nomenclature);
        strcpy(CompVars[NumCompVars].class, "msid");
        CompVars[NumCompVars].occurrence = 1;
        CompVars[NumCompVars].put_or_get = GET;
        CompVars[NumCompVars].lo1_limit = 0;
        CompVars[NumCompVars].lo2_limit = 25;
        CompVars[NumCompVars].hi1_limit = 50;
    }
}
```

90/06/07
13:14:02

put_msid.c

3

```
CompVars[NumCompVars].hi2_limit = 100;  
NumCompVars++;  
}
```

```
/*  
if (LHS) we are setting the type for CompareType  
*/  
if (Equation == LHS)  
    strcpy(CompareType[NumberOfCompares++], type);  
  
/*  
Clean up indentation and add msid to comp  
*/  
indent (PREMISE);  
strcat (Comp, msid);  
updateWA( msid );  
PrevChoice[ChoiceCounter++] = MSID;  
NeedToSave = TRUE;  
}
```


90/06/07
13:14:04

put_signal.c

1

PUT_SIGNAL

Purpose: Put_signal handles the addition of signals into the comp, this includes setting flags and arrays.

Designer: Terri Murphy

Programmer: Terri Murphy

Date: 10/88

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/

/*****
    Include files
*****/
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"

put_signal ()
{
    char    signal[SIGNAL_NAME_LEN], /* A string to get the signal into */
           nomenclature[NOMEN_LEN], /* A string to get the nomenclature into */
           message[155]; /* A place to put your messages */

    int rv1, rv2, /* Return values for var_check() */
        is_a_getter, /* TRUE if we are just looking at the signal */
        var_check(), /* Returns index of inputted signal */
        local_index, /* local index to be set by var_check() */
        global_index; /* global index to be set by var_check() */

    /*****
    We need to know if we are putting or getting to
    determine if the signal is valid.
    *****/
    if ((WhereAmI == CONSEQUENCE || (NumberOfIfs - NumberOfEndifs) == 0) && (Equation == LHS))
        is_a_getter = FALSE;
    else
        is_a_getter = TRUE;
    /*****
    Keep getting variables until we get a valid one
    *****/
    do
    {
        /*
        * Get the name the signal from the user
        */
        if (getVarName (signal, SIGNAL) == ERROR)
            return ERROR;

        if ((rv2 = var_check (signal, &local_index, &global_index, "signal")) == ERROR)
        {
            sprintf(message, "\n\t%s has been previously defined as a LOCAL variable", signal);
            user_ack(message, HELP_U_ACK);
        }
        /*****
        If the signal is in the signal table we should show the
        nomenclature.
        *****/
        if (global_index != -1)
        {
            sprintf(message, "\n\t%s", SignalTable[global_index].nomenclature);
        }
    }
    while (is_a_getter == TRUE);
}
```

```
user_ack( message, HELP_U_ACK );
}
/*****
Check to see that the type comparison is correct
*****/
if (Equation == RHS && global_index != -1 && type_check(SignalTable[global_index].type[0]) == ERROR)
{
    sprintf (message, "\n\t%s is of wrong type. Should have been of type '%s'", signal, SignalTable[global_index].type);
    user_ack(message, HELP_U_ACK);
    rv2 = ERROR;
}
if (Equation == RHS && local_index != -1 && type_check(CompVars[local_index].type[0]) == ERROR)
{
    sprintf (message, "\n\t%s is of wrong type. Should have been of type '%s'", signal, SignalTable[global_index].type);
    user_ack(message, HELP_U_ACK);
    rv2 = ERROR;
}
} while (rv1 == ERROR || rv2 == ERROR);
/*****
rv2 = 0
local_index = -1
global_index = <valid_index>
This means that the signal already exists in the signal table, but it
hasn't been defined locally yet. Create the CompVars entry.
*****/
if (local_index == -1 && global_index != -1)
{
    strcpy(CompVars[NumCompVars].name, signal);
    CompVars[NumCompVars].occurrence = 1;
    strcpy(CompVars[NumCompVars].class, "signal");
    if (is_a_getter)
        CompVars[NumCompVars].put_or_get = GET;
    else
        CompVars[NumCompVars].put_or_get = PUT;
    CompVars[NumCompVars].lo1_limit = 0;
    CompVars[NumCompVars].lo2_limit = 25;
    CompVars[NumCompVars].hi1_limit = 50;
    CompVars[NumCompVars].hi2_limit = 100;
    strcpy(CompVars[NumCompVars].nomenclature,
        SignalTable[global_index].nomenclature);
    strcpy(CompVars[NumCompVars].type, SignalTable[global_index].type);

    /*****
    if (LHS) we are setting the type for this compare
    *****/
    if (Equation == LHS)
        strcpy (CompareType[NumberOfCompares++], SignalTable[global_index].type);
    NumCompVars++;
}
/*****
rv2 = 0
local_index != -1
Means that the signal is already defined locally.
Update the occurrence count, make sure the
the signal is still being used in the same way.
*****/
else if (local_index != -1)
{
    /*****
    Let's make sure that if they were getting they're
    still getting - or if they're putting now we need
    to change put_or_get to "PET" (and vice versa).
    *****/
    if ((is_a_getter && CompVars[local_index].put_or_get == PUT) ||
        (!is_a_getter && CompVars[local_index].put_or_get == GET))
        CompVars[local_index].put_or_get = PET;
    /*****
    Increment the occurrence of this signal
    *****/
    CompVars[local_index].occurrence++;
    /*****
    if (LHS) we are setting the type for this compare
    *****/
    if (Equation == LHS)
        strcpy (CompareType[NumberOfCompares++], CompVars[local_index].type);
}
}
```

90/06/07
13:14:04

put_signal.c

3

```

/*****
If the rv2 is 0
local_index is -1
global_index is -1
The signal is not defined anywhere - SO DEFINE IT!!!
*****/
else if (local_index == -1 && global_index == -1)
{
    strcpy(CompVars[NumCompVars].name , signal);
    CompVars[NumCompVars].occurrence = 1;
    strcpy(CompVars[NumCompVars].class, "signal");
    CompVars[NumCompVars].put_or_get = PUT;
    CompVars[NumCompVars].lo1_limit = 0;
    CompVars[NumCompVars].lo2_limit = 25;
    CompVars[NumCompVars].hi1_limit = 50;
    CompVars[NumCompVars].hi2_limit = 100;
    get_string("Please enter signal nomenclature.",nomenclature,NOMEN_LEN-1,FALSE);
    sprintf(CompVars[NumCompVars].nomenclature, " /* %s */", nomenclature);
    /*****/
    If LHS then we get the type set the CompareType to it
    and increment the NumberOfCompares counter
    *****/
    if (Equation == LHS)
    {
        get_type(CompVars[NumCompVars].type, "Of what type?");
        strcpy (CompareType[NumberOfCompares++], CompVars[NumCompVars].type);
    }
    /*****/
    If RHS then we default the signal to the CompareType
    *****/
    else
    {
        strcpy (CompVars[NumCompVars].type, CompareType[NumberOfCompares-1]);
    }

    NumCompVars++;
}
/*****
Clean up indentation and add signal to comp
*****/
indent (PREMISE);
strcat (Comp, signal);
updateWA( signal );
PrevChoice[ChoiceCounter++] = SIGNAL;
NeedToSave = TRUE;
}

```

90/06/07
13:14:06

put_status.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
#include "widgets.h"
```

```
/*----->*****
*
* MODULE NAME:  put_status( void )
*
* MODULE FUNCTION:
*
*   Routine updates the status window of the Comp Builder main screen.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*----->*****/
```

```
int put_status()
{
    char    status[100];    /* The status string */

    sprintf( status, "GROUP: %s    COMP: %s    STATUS: ", GroupName, CompName );

    /*
     * The comp is complete if the length of the comp is greater
     * than 2 and there are an equal number of "if"s and "endif"s.
     */

    if (CompInfo[CompNumber].disposition == ERROR && !NeedToSave)
        strcat(status, "Error");
    else if (CompInfo[CompNumber].disposition == INSTALLED && !NeedToSave)
        strcat(status, "Installed");
    else if (NumberOfIfs-NumberOfEndifs == 0)
    {
        if ( PrevChoice[ChoiceCounter-1] == END_IF ||
            PrevChoice[ChoiceCounter-1] == PRINT ||
            PrevChoice[ChoiceCounter-1] == COMMENT ||
            Equation == RHS &&
            (PrevChoice[ChoiceCounter-1] == MSID ||
            PrevChoice[ChoiceCounter-1] == NUMBER ||
            PrevChoice[ChoiceCounter-1] == STRING ||
            PrevChoice[ChoiceCounter-1] == SIGNAL ||
            PrevChoice[ChoiceCounter-1] == LOCAL ||
            PrevChoice[ChoiceCounter-1] == PI ||
            (PrevChoice[ChoiceCounter-1] == R_PAREN && ParenCount == 0)))
        {
            strcat(status, "Complete");
            CompInfo[CompNumber].disposition = COMPLETE;
        }
        else
        {
            strcat(status, "Incomplete");
            CompInfo[CompNumber].disposition = INCOMPLETE;
        }
    }
    else
    {
        CompInfo[CompNumber].disposition = INCOMPLETE;
        strcat(status, "Incomplete");
    }

    /*
     * Display the "cycle selection mode".
     */

    strcat ( status, "    MODE: " );
    if ( Disposition != NO_GROUP )
        switch ( CompInfo[CompNumber].cycle_mode )
        {
            case CYCLIC      : strcat( status, "CYCLIC" );
                             arm_toggle( tgl_cycle );
                             disarm_toggle( tgl_lshot );
                             break;

            case ONE_SHOT    : strcat( status, "ONE-SHOT" );
                             arm_toggle( tgl_lshot );
                             disarm_toggle( tgl_cycle );
                             break;

            case NONE        : disarm_toggle( tgl_cycle );
                             disarm_toggle( tgl_lshot );
                             break;

            default          : user_ack( "Comp contains invalid Cycle Selection Mode", HELP_U_ACK );
        }

    /*
     * Display the status string.
     */

    XmTextSetString( txt_status, status );

    /*
     * If there is danger of the comp exceeding the MAX_COMP_LEN if
     * the largest token (MAX_COMMENT_LEN) is selected next, tell
     * the user to notify developers and EXIT before irrevocable
     * damage is done.
     */
}
```

90/06/07
13:14:06

put_status.c

3

```
*/  
if ( strlen(Comp) > (MAX_COMP_LEN - MAX_COMMENT_LEN))  
{  
    user_ack("You have exceeded the maximum comp length. Notify developers.", HELP_U_ACK);  
    save( NO_SHOW );  
    cleanExit();  
}
```

```
*****<----->*****
*
* FILE NAME:      put_token.c
*
* FILE FUNCTION:
*
*   Adds the cooresponding token to the active Comp.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* FILE MODULES:
*
*   put_add()           -
*   put_and()           -
*   put_bitAnd()        -
*   put_bitOr()         -
*   put_bitXor()        -
*   put_comma()         -
*   put_comment()       -
*   put_divide()        -
*   put_else()          -
*   put_endif()         -
*   put_eq()            -
*   put_func( choice ) -
*   put_ge()            -
*   put_gt()            -
*   put_if()            -
*   put_l_paren()       -
*   put_le()            -
*   put_lt()            -
*   put_multiply()      -
*   put_ne()            -
*   put_not()           -
*   put_number()        -
*   put_or()            -
*   put_pi()            -
*   put_print()         -
*   put_r_paren()       -
*   put_set()           -
*   put_shiftL()        -
*   put_shiftR()        -
*   put_start()         -
*   put_stop()          -
*   put_string()        -
*   put_subtract()      -
*   put_then()          -
*
*****<----->*****/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"

*****<----->*****
*
* MODULE NAME: put_add()
*
* MODULE FUNCTION:
*
*   Adds the "add" token to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*****<----->*****
```

90/06/07
13:14:09

put_token.c

2

* ORIGINAL AUTHOR AND IDENTIFICATION:

*
* Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16
*

*****<----->*****/

put_add()

```
{  
    strcat( Comp, " +" );  
    updateWA( " +" );  
    NeedToSave = TRUE;  
    PrevChoice[ChoiceCounter++] = ADD;  
}
```

*****<----->*****/

* MODULE NAME: put_and()

* MODULE FUNCTION:

* Adds the "and" token to the current Comp.
*

* SPECIFICATION DOCUMENTS:

* /code/specs/code
*

* ORIGINAL AUTHOR AND IDENTIFICATION:

*
* Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16
*

*****<----->*****/

put_and()

```
{  
  
    strcat( Comp, " and" );  
    updateWA( " and" );  
    Equation = LHS;  
    NeedToSave = TRUE;  
    PrevChoice[ChoiceCounter++] = AND;  
}
```

*****<----->*****/

* MODULE NAME: put_bitAnd()

* MODULE FUNCTION:

* Adds the "bitAnd" token to the current Comp.
*

* SPECIFICATION DOCUMENTS:

* /code/specs/code

90/06/07
13:14:09

put_token.c

3

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

put_bitAnd()

```
{  
    strcat( Comp, " bitAnd" );  
    updateWA( " bitAnd" );  
    NeedToSave = TRUE;  
    PrevChoice[ChoiceCounter++] = BITAND;  
}
```

/*****<----->*****/

* MODULE NAME: put_bitOr()

* MODULE FUNCTION:

* Adds the "bitOr" token to the current Comp.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

put_bitOr()

```
{  
    strcat( Comp, " bitOr" );  
    updateWA( " bitOr" );  
    NeedToSave = TRUE;  
    PrevChoice[ChoiceCounter++] = BITOR;  
}
```

/*****<----->*****/

* MODULE NAME: put_bitXor()

* MODULE FUNCTION:

* Adds the "bitXor" token to the current Comp.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

90/06/07
13:14:09

put_token.c

4

```
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
```

* REVISION HISTORY:

```
*
*   Motif Release 1.0 - 90/03/16
*
```

*****<----->*****/

put_bitXor()

```
{
    strcat( Comp, " bitXor" );
    updateWA( " bitXor" );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = BITXOR;
}
```

90/06/07
13:14:09

put_token.c

5

```

/*****<----->*****/
*
* MODULE NAME: put_comma()
*
*
* MODULE FUNCTION:
*
*   Adds the "comma" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*****/

```

```

put_comma()
{
    strcat( Comp, " ," );
    updateWA( " ," );
    NeedToSave = TRUE;
    FunctionArgsDef[FunctionCurrent]++;
    PrevChoice[ChoiceCounter++] = COMMA;
}

```

```

/*****<----->*****/
*
* MODULE NAME: put_comment()
*
*
* MODULE FUNCTION:
*
*   Adds the "comment" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*****/

```

```

put_comment()
{
    char comment[MAX_COMMENT_LEN+1];
    int rc;

    /*

```

90/06/07
13:14:09

put_token.c

6

```

/*
 * If no group is selected, let the user know to use CREATE.
 */

if ( Disposition == NO_GROUP )
{
    user_ack( "No group selected, use CREATE to start a new Group/Comp", HELP_U_ACK );
    return;
}

/*
 * Loop until the user enters a comment string.
 */

do {
    comment[0] = NULL;
    rc = get_string( "Please enter your comment below", comment, MAX_COMMENT_LEN, TRUE );
    if ( rc == ABORT )
        user_ack("Please enter a valid comment", HELP_U_ACK);
} while ( rc == ABORT );

strcat( Comp, " /* " );
strcat( Comp, comment );
strcat( Comp, " */" );
updateWA( " /* " );
updateWA( comment );
updateWA( " */" );
NeedToSave = TRUE;
PrevChoice[ChoiceCounter++] = COMMENT;
}

/*****<----->*****/
*
* MODULE NAME: put_divide()
*
*
* MODULE FUNCTION:
*
* Adds the "divide" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
* Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

put_divide()
{
    strcat( Comp, " /" );
    updateWA( " /" );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = DIVIDE;
}

/*****<----->*****/
*
* MODULE NAME: put_else()
*
*
* MODULE FUNCTION:
*
* Adds the "else" token to the current Comp.
*
```

90/06/07
13:14:09

put_token.c

7

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD

* Timothy J. Barton - Software Engineering Section
Data Systems Department
Automation and Data Systems Division
Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

put_else()

```
(
    int i;

    NestedElseCheck[ NumberOfIfs - NumberOfEndifs ] = ELSE;
    strcat( Comp, "\n" );
    updateWA( "\n" );
    for (i=0; i < ((NumberOfIfs-NumberOfEndifs)-1)*SIZE_INDENT; i++ )
    {
        strcat( Comp, " " );
        updateWA( " " );
    }
    strcat ( Comp, "else" );
    updateWA( "else" );
    Equation = LHS;
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = ELSE;
)
```

*****<----->*****/

* MODULE NAME: put_endif()

* MODULE FUNCTION:

* Adds the "endif" token to the current Comp.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD

* Timothy J. Barton - Software Engineering Section
Data Systems Department
Automation and Data Systems Division
Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

put_endif()

```
(
    NumberOfEndifs++;
    indent( CONSEQUENCE );
    strcat( Comp, "endif" );
    updateWA( "endif" );
)
```

```

NeedToSave = TRUE;
PrevChoice[ChoiceCounter++] = END_IF;
}

/*****<----->*****/
*
* MODULE NAME: put_eq()
*
*
* MODULE FUNCTION:
*
*   Adds the "equal" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
put_eq()
{
    if (WhereAmI == CONSEQUENCE || NumberOfifs == NumberOfEndifs)
    {
        strcat( Comp, " =" );
        updateWA( " =" );
    }
    else /* we're in the premise */
    {
        strcat( Comp, " ==" );
        updateWA( " ==" );
    }

    PrevChoice[ChoiceCounter++] = EQ;
    Equation = RHS;
    NeedToSave = TRUE;
}

/*****<----->*****/
*
* MODULE NAME: put_func( choice )
*
*
* MODULE FUNCTION:
*
*   Adds the "function" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute

```

90/06/07
13:14:09

put_token.c

9

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

put_func(choice)

int choice;

```
{
    int    rc;                                /* Fuction call return code */
    char    message[150],
            whichFunction[ FUNC_NAME_LEN ];    /* The actual name of the selected function */

    switch(choice)
    {
        case COS:    strcpy(FunctionList[FunctionCount], "cos");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case ACOS:    strcpy(FunctionList[FunctionCount], "acos");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case SIN:     strcpy(FunctionList[FunctionCount], "sin");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case ASIN:    strcpy(FunctionList[FunctionCount], "asin");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case TAN:     strcpy(FunctionList[FunctionCount], "tan");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case ATAN:    strcpy(FunctionList[FunctionCount], "atan");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case SQRT:    strcpy(FunctionList[FunctionCount], "sqrt");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case POWER:   strcpy(FunctionList[FunctionCount], "power");
                     FunctionArguments[FunctionCount] = 2;
                     break;

        case LOG:     strcpy(FunctionList[FunctionCount], "log");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case EXP:     strcpy(FunctionList[FunctionCount], "exp");
                     FunctionArguments[FunctionCount] = 1;
                     break;

        case FUNCTION:
            do {
                strcpy( message, "Please enter the function name below" );
                if ( get_string(message,whichFunction,FUNC_NAME_LEN-1,FALSE) == ABORT )
                    return( DEFAULT );
                if ( (rc = funcCheck(whichFunction)) == ERROR )
                {
                    sprintf ( message, " '%s' is not a defined user function. ", whichFunction );
                    user_ack( message, HELP_U_ACK );
                }
            } while ( rc == ERROR );

            strcpy( FunctionList[ FunctionCount ], whichFunction );
            FunctionArguments[ FunctionCount ] = MAX_FUNC_PARAMS;
            break;

        default:      break;
    }

    FunctionCurrent = FunctionCount;
}
```

90/06/07
13:14:09

put_token.c

10

```
FunctionArgsDef[ FunctionCount ] = 0;

/*
 * Clean up indentation
 */

indent( PREMISE );
strcat(Comp, FunctionList[FunctionCount++] );
updateWA( FunctionList[FunctionCount-1] );
PrevChoice[ChoiceCounter++] = choice;
put_l_paren ();

/*
 * We're now in a function so increment the function
 * paren count if it wasn't already incremented by put_l_paren.
 * (It will be incremented in put_l_paren if it's a nested
 * function.)
 */

if( FuncParenCount == 0 )
    FuncParenCount++;
NeedToSave = TRUE;
}

/*****<----->*****/
*
* MODULE NAME: put_ge()
*
*
* MODULE FUNCTION:
*
* Adds the "greater-equal" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
* Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

put_ge()
{
    strcat( Comp, " >=" );
    updateWA( " >=" );
    PrevChoice[ChoiceCounter++] = GE;
    Equation = RHS;
    NeedToSave = TRUE;
}

/*****<----->*****/
*
* MODULE NAME: put_gt()
*
*
* MODULE FUNCTION:
*
* Adds the "greater-than" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
```


90/06/07
13:14:09

put_token.c

11

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

put_gt()

```
{
    strcat( Comp, " >" );
    updateWA( " >" );
    PrevChoice[ChoiceCounter++] = GT;
    Equation = RHS;
    NeedToSave = TRUE;
}
```

*****<----->*****

* MODULE NAME: put_if()

* MODULE FUNCTION:

* Adds the "if" token to the current Comp.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

*****<----->*****/

put_if()

```
{
    /*
     * If we're balanced then we just add a return
     */

    if ((NumberOfIfs - NumberOfEndifs) != 0)
        indent( CONSEQUENCE );
    else
    {
        strcat( Comp, "\n" );
        updateWA( "\n" );
    }

    strcat( Comp, "if" );
    updateWA( "if" );
    NumberOfIfs++;
    PrevChoice[ChoiceCounter++] = IF;
    WhereAmI = PREMISE;
    Equation = LHS;
    NeedToSave = TRUE;
}
```

90/06/07
13:14:09

put_token.c

12

```

/*****<----->*****/
*
* MODULE NAME: put_l_paren()
*
* MODULE FUNCTION:
*   Adds the "left paren" token to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*   Motif Release 1.0 - 90/03/16
*****/
```

```

put_l_paren()
{
    strcat( Comp, " ( " );
    updateWA( " ( " );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = L_PAREN;

    /*
     * If we're defining function arguments we need to increment the function
     * paren count so we can tell when we've finished arg def'ing.
     */

    if( FuncParenCount > 0 )
        FuncParenCount++;
    ParenCount++;
}

```

```

/*****<----->*****/
*
* MODULE NAME: put_le()
*
* MODULE FUNCTION:
*   Adds the "less than equal" token to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*   Motif Release 1.0 - 90/03/16
*
```

90/06/07
13:14:09

put_token.c

13

put_le()

```
{
    strcat( Comp, " <=" );
    updateWA( " <=" );
    PrevChoice[ChoiceCounter++] = LE;
    Equation = RHS;
    NeedToSave = TRUE;
}
```

/*****<----->*****/

* MODULE NAME: put_lt()

* MODULE FUNCTION:

* Adds the "less than" token to the current Comp.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD

* Timothy J. Barton - Software Engineering Section
Data Systems Department
Automation and Data Systems Division
Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

/*****<----->*****/

put_lt()

```
{
    strcat( Comp, " <" );
    updateWA( " <" );
    PrevChoice[ChoiceCounter++] = LT;
    Equation = RHS;
    NeedToSave = TRUE;
}
```

/*****<----->*****/

* MODULE NAME: put_multiply

* MODULE FUNCTION:

* Adds the "multiply" token to the current Comp.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD

* Timothy J. Barton - Software Engineering Section
Data Systems Department
Automation and Data Systems Division
Southwest Research Institute

* REVISION HISTORY:

90/06/07
13:14:09

put_token.c

14

```
* Motif Release 1.0 - 90/03/16
*
*****<----->*****/

put_multiply()
{
    strcat( Comp, " *" );
    updateWA( " *" );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = MULTIPLY;
}

/*****<----->*****
*
* MODULE NAME: put_ne()
*
*
* MODULE FUNCTION:
*
*   Adds the "not equal" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****<----->*****/

put_ne()
{
    strcat( Comp, " 8" );
    updateWA( " 8" );
    PrevChoice[ChoiceCounter++] = NE;
    Equation = RHS;
    NeedToSave = TRUE;
}

/*****<----->*****
*
* MODULE NAME: put_not()
*
*
* MODULE FUNCTION:
*
*   Adds the "not" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
```

```
*
* Motif Release 1.0 - 90/03/16
*
*****<----->*****/

put_not()
{
    indent( PREMISE );
    strcat( Comp, "not" );
    updateWA( "not" );
    PrevChoice[ChoiceCounter++] = NOT;
    put_l_paren();
    NeedToSave = TRUE;
}

/*****<----->*****/
*
* MODULE NAME: put_number()
*
*
* MODULE FUNCTION:
*
* Adds a number to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****<----->*****/

put_number()
{
    char number[30];
    int status;

    /*
     * Loop until the user gives us a 'good' number
     */

    do {
        status = get_string( "Please enter a valid number below", number, 30-1, FALSE );
        if ( status == ABORT )
        {
            user_ack( "Please enter a valid number", HELP_U_ACK );
            status = ERROR;
        }
        else
            if ((status = str_isalnum(number)) == ERROR )
                user_ack("Invalid number", HELP_U_ACK);

    } while ( status == ERROR );

    if ( Equation == LHS )
        if ( status == FLOAT )
            strcpy( CompareType[ NumberOfCompares++ ], "float" );
        else
            strcpy( CompareType[ NumberOfCompares++ ], "int" );

    /*
     * Clean up indentation
     */
}
```

90/06/07
13:14:09

put_token.c

16

```
indent( PREMISE );

strcat( Comp, number );
updateWA( number );
NeedToSave = TRUE;
PrevChoice[ChoiceCounter++] = NUMBER;
}

/*****<----->*****/
*
* MODULE NAME: put_or()
*
* MODULE FUNCTION:
*
*   Adds the "or" token to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

put_or()
{
    strcat( Comp, " or" );
    updateWA( " or" );
    Equation   = LHS;
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = OR;
}

/*****<----->*****/
*
* MODULE NAME: put_pi()
*
* MODULE FUNCTION:
*
*   Adds the "PI" token to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

```
put_pi()
(
    indent( PREMISE );
    strcat( Comp, "PI" );
    updateWA( "PI" );
    PrevChoice[ChoiceCounter++] = PI;
    NeedToSave = TRUE;
)

/*****<----->*****/
*
* MODULE NAME: put_print()
*
*
* MODULE FUNCTION:
*
* Adds a print statement to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
* Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

put_print()
(
    char fault[82],
        prompt[100];
    int rc;

    indent( CONSEQUENCE );
    strcat( Comp, "print" );
    updateWA( "print" );
    NeedToSave = TRUE;

    /*
     * Get the fault msg class from the user
     */

    do
    (
        fault[0] = NULL;
        rc = get_string("Please enter the fault message class (1-5).", fault, 1, FALSE );
        if ( rc == ABORT )
            user_ack("Please enter a fault message class", HELP_U_ACK);
    )
    while (rc == ABORT || atoi(fault) < 1 || atoi(fault) > 5);

    /*
     * Append the flt msg class to comp
     */

    strcat( Comp, fault );
    updateWA( fault );
    strcat( Comp, " \042" );
    updateWA( " \042" );

    /*
     * Get the fault msg from the user
     */
}
```

90/06/07
13:14:09

put_token.c

18

```
strcpy( prompt, "Please enter fault message. Precede variables by percent" );
strcat( prompt, "\n\tsgn. (e.g. V75T2517A is %V75T2517A)" );

do {
    fault[0] = NULL;
    rc = get_string( prompt, fault, 81, FALSE );
    if ( rc == ABORT )
        user_ack("Please enter the fault message", HELP_U_ACK);
} while( rc == ABORT );

/*
 * Append the fault msg to the comp
 */

strcat( Comp, fault );
strcat( Comp, "\042" );
updateWA( fault );
updateWA( "\042" );
PrevChoice[ChoiceCounter++] = PRINT;
}

/*****<----->*****/
*
* MODULE NAME: put_r_paren()
*
*
* MODULE FUNCTION:
*
*   Adds a "right paren" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

put_r_paren()
{
    int i;

    strcat( Comp, " )" );
    updateWA( " )" );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = R_PAREN;

    /*
     * If we're defining function arguments
     * we need to decrement the function
     * paren count so we can tell when we've
     * finished arg def'ing.
     */

    if( FuncParenCount > 0 )
    {
        FuncParenCount--;
        if ( ++FunctionArgsDef[ FunctionCurrent ] == FunctionArguments[ FunctionCurrent ] )
        {
            for ( i=FunctionCurrent-1; i >= 0; i--)
            {
                if ( FunctionArgsDef[i] != FunctionArguments[i] )
                {
                    FunctionCurrent = i;
                }
            }
        }
    }
}
```


90/06/07
13:14:09

put_token.c

19

```
        break;
    }
}

    }
    ParenCount--;
}

/*****<----->*****/
*
* MODULE NAME:  put_set()
*
* MODULE FUNCTION:
*
*   Adds a set statement to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
put_set()
{
    indent( CONSEQUENCE );
    strcat( Comp, "set" );
    updateWA( "set" );
    Equation   = LHS;
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = SET;
}

/*****<----->*****/
*
* MODULE NAME:  put_shiftL()
*
* MODULE FUNCTION:
*
*   Adds a "shift left" token to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
```

90/06/07
13:14:09

put_token.c

20

```
put_shiftL()
```

```
{
    strcat( Comp, " shiftL" );
    updateWA( " shiftL" );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = SHIFTL;
}
```

```
/*****<----->*****/
```

```
* MODULE NAME: put_shiftR()
```

```
* MODULE FUNCTION:
```

```
* Adds a "shift right" token to the current Comp.
```

```
* SPECIFICATION DOCUMENTS:
```

```
* /code/specs/code
```

```
* ORIGINAL AUTHOR AND IDENTIFICATION:
```

```
* Troy Heindel & Terri Murphy of NASA/JSC/MOD
```

```
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
```

```
* REVISION HISTORY:
```

```
* Motif Release 1.0 - 90/03/16
```

```
****<----->*****/
```

```
put_shiftR()
```

```
{
    strcat( Comp, " shiftR" );
    updateWA( " shiftR" );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = SHIFTR;
}
```

```
/*****<----->*****/
```

```
* MODULE NAME: put_start()
```

```
* MODULE FUNCTION:
```

```
* Adds a "start" token to the current Comp.
```

```
* SPECIFICATION DOCUMENTS:
```

```
* /code/specs/code
```

```
* ORIGINAL AUTHOR AND IDENTIFICATION:
```

```
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
```

```
* REVISION HISTORY:
```

```
* Motif Release 1.0 - 90/03/16
```

```
****<----->*****/
```

90/06/07
13:14:09

put_token.c

21

```
put_start()
{
    char    comp_name[100];
    int     rc;

    indent( CONSEQUENCE );
    PrevChoice[ChoiceCounter++] = START;
    strcat( Comp, "start( " );
    updateWA( "start( " );

    rc = get_string("Please enter the comp name.", comp_name, 100-1, FALSE );
    if ( rc == ABORT )
    {
        delete();
        displayWA( Comp );
        return;
    }
    else
    {
        strcat( Comp, comp_name );
        strcat( Comp, " )" );
        updateWA( comp_name );
        updateWA( " )" );
    }

    NeedToSave = TRUE;
}

/*****<----->*****/
*
* MODULE NAME: put_stop()
*
*
* MODULE FUNCTION:
*
* Adds a "stop" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
*                   Data Systems Department
*                   Automation and Data Systems Division
*                   Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

put_stop()
{
    char    comp_name[100];
    int     rc;

    indent( CONSEQUENCE );
    PrevChoice[ChoiceCounter++] = STOP;
    strcat( Comp, "stop( " );
    updateWA( "stop( " );

    rc = get_string("Please enter the comp name.", comp_name, 100-1, FALSE );
    if ( rc == ABORT )
    {
        delete();
        displayWA( Comp );
        return;
    }
    else
    {
        strcat( Comp, comp_name );
        strcat( Comp, " )" );
    }
}
```

90/06/07
13:14:09

put_token.c

22

```
        updateWA( comp_name );
        updateWA( " " );
    )

    NeedToSave = TRUE;
}

/*****<----->*****/
*
* MODULE NAME: put_string()
*
* MODULE FUNCTION:
*
*   Adds a string to the current Comp.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
put_string()
{
    char    theString1[ MAX_STR_LEN ],
            theString2[ MAX_STR_LEN ];
    int     rc;

    theString1[0] = NULL;
    theString2[0] = NULL;

    /*
     * Get the string from the user
     */

    do {
        rc = get_string("Please enter your text string below", theString1, MAX_STR_LEN-1, FALSE );
        if ( rc == ABORT )
            user_ack("Please enter a text string", HELP_U_ACK);
    } while ( rc == ABORT );

    if ( Equation == LHS )
        strcpy( CompareType[ NumberOfCompares++ ], "char" );

    /*
     * Clean up indentation
     */

    indent( PREMISE );

    /*
     * Append the string to the comp
     */

    sprintf( theString2, "%042s%042s", theString1 );
    strcat ( Comp, theString2 );
    updateWA( theString2 );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = STRING;
}

/*****<----->*****/
```

90/06/07
13:14:09

put_token.c

23

```
*
* MODULE NAME: put_subtract()
*
*
* MODULE FUNCTION:
*
*   Adds a "subtract" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****<----->*****/

put_subtract()
{
    strcat( Comp, " -" );
    updateWA( " -" );
    NeedToSave = TRUE;
    PrevChoice[ChoiceCounter++] = SUBTRACT;
}

/*****<----->*****/
*
* MODULE NAME: put_then()
*
*
* MODULE FUNCTION:
*
*   Adds a "then" token to the current Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel & Terri Murphy of NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****<----->*****/

put_then()
{
    int i;

    NestedElseCheck[NumberOfIfs-NumberOfEndifs] = THEN;

    /*
     *   Clean up indentation
     */
}
```

```
strcat( Comp, "\n" );
updateWA( "\n" );
for (i=0; i < ((NumberOfIfs-NumberOfEndifs)-1)*SIZE_INDENT; i++ )
{
    strcat( Comp, " " );
    updateWA( " " );
}
strcat( Comp, "then" );
updateWA( "then" );
WhereAmI = CONSEQUENCE;
Equation = LHS;
NeedToSave = TRUE;
PrevChoice[ChoiceCounter++] = THEN;
}
```

90/06/07
13:14:13

remove.c

1

REMOVE

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 1/27/88

Version: 1.0

Project: CODE (Comp Development Environment)
IESP (INCO Expert System Project)

Revised by:

Reasons for Revision:

External Interfaces

*****/

```
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"
```

/*-----*/

* MODULE NAME: remove()

* MODULE FUNCTION:

* Determines if the user wants to delete a group or comp, then determines which
* group or comp to delete.

* SPECIFICATION DOCUMENTS:

* /code/specs/code

* ORIGINAL AUTHOR AND IDENTIFICATION:

* Troy Heindel & Terri Murphy of NASA/JSC/MOD

* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute

* REVISION HISTORY:

* Motif Release 1.0 - 90/03/16

/*-----*/

remove ()

```
{
    FILE *ptr1;
    int is_installed, /* TRUE if the comp being removed was prev. installed, else FALSE */
        i,j,
        rc;
```

```
    char group_dat_file[PATH_LEN]; /* CompInfo file name */
    char confirm[250], cmd_string[250];
```

```
/*
 * Ask the user if they want to delete an entire group or just
 * one comp.
 */
```

```
rc = get_type_gc( "Do you want to remove a Group or Comp?" );
if ( rc == ABORT )
    return( OK );

/*
 * User chose to delete a comp.
 */

if ( rc == COMP )
{
    rc = get_name (NumOfGroups, "Group", GroupName, &GroupNumber, &Disposition);
    if (rc == ERROR)
        return(ERROR);
    rc = get_name (NumOfComps, "Comp", CompName, &CompNumber, &Disposition);
    if (rc == ERROR)
        return(ERROR);
    retrieve();

    /*
     * Give the user one more chance to abort this drastic action.
     */

    displayWA(Comp);
    sprintf(confirm, "Are you sure you want to remove %s", CompName);
    if (! ask(confirm) )
        return;

    select_cursor( Clock_Cursor );

    /*
     * Remove the comp from the <group>.dat file.
     */

    for(i=0;i<NumOfComps;i++)
    {
        if(strcmp(CompName, CompInfo[i].name) == 0)
        {
            if(CompInfo[i].disposition == INSTALLED)
                is_installed = TRUE;
            else
                is_installed = FALSE;
            for (j=i;j<NumOfComps;j++)
                CompInfo[j] = CompInfo[j+1];
            break;
        }
    }

    NumOfComps--;

    /*
     * Save the CompInfo.
     */

    sprintf(group_dat_file, "%s/%s.dat", AMSupport, GroupName);
    if (!(ptr1 = fopen(group_dat_file, "w")))
    {
        select_cursor( Shuttle_Cursor );
        sprintf(cmd_string, "You do not have r/w permission to save %s.dat", GroupName);
        user_ack( cmd_string, HELP_U_ACK );
        return( ERROR );
    }

    fprintf(ptr1, "%s\n", "#name_name noise_filter rate on_off disposition cycle_mode\n");
    for (i=0;i < NumOfComps;i++)
    {
        fprintf (ptr1,"%s %d %d %d %d %s\n",
            CompInfo[i].name,
            CompInfo[i].noise_filter,
            CompInfo[i].rate,
            CompInfo[i].on_off,
            CompInfo[i].disposition,
            CompInfo[i].cycle_mode,
            CompInfo[i].purpose);
    }
    fclose( ptr1 );

    /*
     * Remove the high_level, c, and variable file from the group directory.
     */
}
```



```
*/  
  
sprintf(cmd_string, "rm -f %s/%s/%s.* 2>>/tmp/code.err", CodeGroups, GroupName, CompName);  
system (cmd_string);  
sprintf(cmd_string, "Comp %s has been removed", CompName);  
sleep(2);  
  
/*  
 * If the comp was previously installed, we need to  
 * install the group again using the new '.dat' file.  
 */  
  
select_cursor( Shuttle_Cursor );  
  
if (is_installed)  
    install();  
  
}  
  
/*  
 * User chose to delete a group.  
 */  
  
else if ( rc == GROUP )  
{  
    rc = get_name (NumOfGroups, "Group", GroupName, &GroupNumber, &Disposition);  
    if (rc == ERROR)  
        return( ERROR );  
  
    /*  
     * Give the user one more chance to abort this action.  
     */  
  
    sprintf(confirm, "Are you sure you want to remove %s", GroupName);  
    if (! ask(confirm) )  
        return;  
  
    select_cursor( Clock_Cursor );  
  
    /*  
     * So far so good --- remove that group and it's .dat  
     * and the executable.  
     */  
  
    sprintf (cmd_string, "rm -r %s/%s %s/%s.dat %s/%s 2>>/tmp/code.err", CodeGroups, GroupName,  
              AMSupport, GroupName, AMGroups, GroupName);  
    system (cmd_string);  
  
    /*  
     * Remove the bugger from the group_names list by sliding everybody  
     * else up a notch.  
     */  
  
    for (i=0; i<NumOfGroups; i++)  
    {  
        if (!strcmp(GroupName, GroupInfo[i].name))  
        {  
            for(j=i; j<NumOfGroups-1; j++)  
            {  
                GroupInfo[j].disposition = GroupInfo[j+1].disposition;  
                strcpy(GroupInfo[j].name, GroupInfo[j+1].name);  
            }  
            break;  
        }  
    }  
  
    NumOfGroups--; /* Take away one for the one just dropped */  
  
    /*  
     * Write back the changes to the GroupNames file  
     */  
  
    writeGroupNames();  
  
    select_cursor( Shuttle_Cursor );  
  
    sprintf(confirm, "Group %s has been deleted", GroupName);  
    user_ack( confirm, HELP_U_ACK );
```

90/06/07
13:14:13

remove.c

4

```
/*  
 * Since we've just deleted what's in the work area let's do  
 * some house cleaning.  
 */
```

```
    CompName[0] = NULL;  
    GroupName[0] = NULL;  
    NeedToSave  = FALSE;  
    NumOfComps  = 0;  
    Disposition = NO_GROUP;  
    Cycle_Mode  = NONE;
```

```
    cleanSlate();  
    displayWA( Comp );  
    put_status();
```

```
/*  
 * Set up the previous choice so the call to color valid  
 * in code.c will only allow the choices we allow on startup.  
 */
```

```
    PrevChoice[ChoiceCounter++] = 99;
```

```
}
```

90/06/07
13:14:16

retrieve.c

1

RETRIEVE

Purpose: Retrieve reads in the comp variable file into the
CompVars struct, reads the comp language file into
a temporary string which is passed to pretty_comp()
for formatting and history set-up.

Returns: 0 - OK, no problems.
-1 - File i/o problem.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/10/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

Include files
*****/
#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"

retrieve ()

```
(
FILE      *ptr, *ptr2; /* File pointers, what else could they be */

char      msg_string[MAX_MSG_LEN], /* Message string repository, and temp string holder */
tempComp[MAX_COMP_LEN], /* Used for temp storage of the comp file */
variable_file[PATH_LEN], /* The string for the path of the comps variable file */
sig_tbl_type[TYPE_LEN], /* Temporary place for signal table type */
header[80]; /* Holds the strings that comprise the signal table header. */

int return_value; /* Return value from fscanf for stripping signal table header */

/*****
Initialize some global variables
*****/
cleanSlate();
SignalCount = 0;

/*
 * Load the signal list and info for signal validation in put_signal.c
 */
if(ptr = fopen (SignalTbl,"r"))
{
    /*
     * Rip the header off the file
     */
    do
    {
        return_value = fscanf (ptr, "%s", header);
    } while (!(return_value == EOF || strcmp(header, FIRST_SIGNAL) == 0));
    if(return_value != EOF)
    {
        strcpy(SignalTable[SignalCount].name, header);
        fscanf (ptr, "%s", sig_tbl_type);
        if (sig_tbl_type[0] == 'S')
            strcpy(SignalTable[SignalCount].type, "c");
        else strcpy(SignalTable[SignalCount].type, sig_tbl_type);
        fscanf(ptr, "%[^\n]", SignalTable[SignalCount].nomenclature);
        SignalCount++;
        /*****
        Loop through the SignalTable file reading the first
        three fields.
        *****/
        while ((fscanf (ptr, "%s", SignalTable[SignalCount].name)) != EOF)
        {
            /*****
            Get the signal table type. If it's a string the
            signal table will have an 'S' - CODE expects a 'char'
            *****/
            fscanf (ptr, "%s", sig_tbl_type);
            if (sig_tbl_type[0] == 'S')
                strcpy(SignalTable[SignalCount].type, "char");
            else strcpy(SignalTable[SignalCount].type, sig_tbl_type);
            /*****
            Get the nomenclature.
            *****/
            fscanf(ptr, "%[^\n]", SignalTable[SignalCount].nomenclature);
            ++SignalCount;
        }
    }
    fclose (ptr);
}

/*****
Let's check for the comp var file and read them if they exist.
*****/
sprintf (variable_file, "%s/%s/%s.v", CodeGroups, GroupName, CompName);
if (!(ptr2 = fopen (variable_file, "r")))
{
    user_ack("Variables have not been found for this comp.", HELP_U_ACK);
    return(ERROR);
}
else
{
    /*****
    First we will discard the comment
    that is the 1st line of all variable files.
    *****/
}
```

```
fscanf(ptr2, "%*[^\\n]");
NumCompVars = 0;
/*****
Now let's read the variable info until
we reach the end of the file.
*****/
while(fscanf(ptr2,"%s %s %s %d %d %f %f %f %f %[\\n]",
            CompVars[NumCompVars].name,
            CompVars[NumCompVars].type,
            CompVars[NumCompVars].class,
            &CompVars[NumCompVars].occurrence,
            &CompVars[NumCompVars].put_or_get,
            &CompVars[NumCompVars].lo1_limit,
            &CompVars[NumCompVars].lo2_limit,
            &CompVars[NumCompVars].hi1_limit,
            &CompVars[NumCompVars].hi2_limit,
            CompVars[NumCompVars].nomenclature) != EOF)
{
    NumCompVars++;
}
fclose(ptr2);
}

/*
 * Read the comp into the global string Comp
 */
if (readCODEFile(CompName, tempComp) == ERROR)
{
    user_ack("There was an error reading the comp.", HELP_U_ACK);
    return (ERROR);
}

/*
 * Reformat the comp with pretty_comp and
 * set up the choice history
 */
if (pretty_comp(tempComp) == ERROR)
{
    sprintf(msg_string, "Undefined tokens; Corrections must occur before installation is possible..");
    user_ack(msg_string, HELP_U_ACK);
    CompInfo[CompNumber].disposition = ERROR;
    save();
    return (ERROR);
}
NeedToSave = FALSE;
return (OK);
} /* End of Retrieve */
```

```
*****  
PRETTY_COMP
```

Purpose: Pretty_comp takes a temp_comp string from retrieve and formats it nicely. It also sets up the ChoiceCounter integer array which is a token history.

NOTE: This routine treats the comp file like a script and recreates the comp in the screen the same way as if a user were mousing tokens off the screen. The exceptions are: prints, comments, variables, numbers.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/26/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****  
pretty_comp (rawComp)  
  char *rawComp; /* The string to fix and put into global comp */  
{  
  int    rawIndex, /* The index used with the passed string */  
        i, /* A simple index */  
        local_index, /* A local index for a variable set by var_check */  
        global_index, /* A global index for a variable set by var_check */  
        tokenIndex; /* The index used with the token string */  
  
  char    token[1000]; /* Temporary holding string for a token */  
  
  /*****  
  Loop through the CODE language string, translating  
  tokens to C, until the end of the string is reached.  
  *****/  
  NumberOfIfs = rawIndex = 0;  
  NumberOfEndifs = 0;  
  /*****  
  Read tokens until you run out ...  
  *****/  
  while (rawComp[rawIndex])  
  {  
    /*****  
    Start with a fresh token index  
    *****/  
    tokenIndex = 0;  
    /*****  
    Yank out the control characters and spaces,  
    we don't care about them spaces and such.  
    *****/  
    while (rawComp[rawIndex] == ' ' || rawComp[rawIndex] == '\t' ||  
           rawComp[rawIndex] == '\n')  
    {  
      token[tokenIndex++] = rawComp[rawIndex++];  
    }  
    token[tokenIndex] = 0;  
    strcat (Comp, token);  
    tokenIndex = 0;  
  
    /*****  
    YANK A TOKEN out of the comp's string. Tokens  
    are delimited by spaces, tabs, or line feeds.  
    *****/  
    while (rawComp[rawIndex] != ' ' && rawComp[rawIndex] != '\t' &&  
           rawComp[rawIndex] != '\n' && rawComp[rawIndex] != 0)  
    {  
      token[tokenIndex++] = rawComp[rawIndex++];  
    }  
  }  
}
```

```
}
token[tokenIndex] = 0;

/*****
If the token is null we've reached the end of
the line and need to stop token processing
*****/
if (!token[0]) return (OK);

/*****
COMMENTS
*****/
if (strcmp(token, "/*", 2) == 0)
{
    /*****
    If this comment is not complete we need to get the rest.
    *****/
    if (token[strlen(token)-1] != '/')
    {
        token[tokenIndex] = rawComp[rawIndex];
        while (!(rawComp[rawIndex-1] == '*' && rawComp[rawIndex] == '/'))
            token[tokenIndex++] = rawComp[rawIndex++];
        token[tokenIndex] = 0;
    }
    /*****
    Now add the comment token to the end of the CODE string.
    *****/
    strcat (Comp, token);
    token[0] = 0;
    PrevChoice[ChoiceCounter++] = COMMENT;
}
/*****
PRINT
*****/
else if (strcmp(token, "print", 5) == 0)
{
    strcat (Comp, token);
    /*****
    Get all the chars delimited by double quotes
    *****/
    tokenIndex = 0; /* Start again for the message */
    token[tokenIndex] = 0; /* null that string */
    for (i=0; i<2; i++)
    {
        do /* Grab chars until the double quote */
        {
            token[tokenIndex++] = rawComp[rawIndex++];
        } while (rawComp[rawIndex-1] != '"');
        token[tokenIndex] = 0;
        /*****
        Let's add it to the Comp string
        *****/
        strcat (Comp, token);
        token[0] = 0;
        PrevChoice[ChoiceCounter++] = PRINT;
    }
}
/*****
STRING
*****/
else if (token[0] == '"')
{
    /*****
    Get rest of the string if need be
    *****/
    if (token[strlen(token)-1] != '"')
    {
        do /* Grab chars until the double quote */
        {
            token[tokenIndex++] = rawComp[rawIndex++];
        } while (rawComp[rawIndex-1] != '"');
        token[tokenIndex] = 0;
        /*****
        Let's add it to the comp string
        *****/
        strcat (Comp, token);
    }
}
```

```
token[0] = 0;
PrevChoice[ChoiceCounter++] = STRING;
}
/*****
    IF
    *****/
else if (strcmp(token,"if") == 0)
{
    strcat (Comp, token);
    WhereAmI = PREMISE;
    Equation = LHS;
    NumberOfIfs++;
    PrevChoice[ChoiceCounter++] = IF;
}
/*****
    ENDIF
    *****/
else if (strcmp(token,"endif") == 0)
{
    strcat (Comp, token);
    NumberOfEndifs++;
    PrevChoice[ChoiceCounter++] = END_IF;
}
/*****
    THEN
    *****/
else if (strcmp(token,"then") == 0)
{
    strcat (Comp, token);
    NestedElseCheck[NumberOfIfs-NumberOfEndifs] = THEN;
    PrevChoice[ChoiceCounter++] = THEN;
    WhereAmI = CONSEQUENCE;
    Equation = LHS;
}
/*****
    ELSE
    *****/
else if (strcmp(token,"else") == 0)
{
    strcat (Comp, token);
    NestedElseCheck[NumberOfIfs-NumberOfEndifs] = ELSE;
    PrevChoice[ChoiceCounter++] = ELSE;
    Equation = LHS;
}
/*****
    OR
    *****/
else if (strcmp(token,"or") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = OR;
    Equation = LHS;
}
/*****
    BITXOR
    *****/
else if (strcmp(token,"bitXor") == 0 || strcmp(token,"xor") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = BITXOR;
}
/*****
    BITOR
    *****/
else if (strcmp(token,"bitOr") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = BITOR;
}
/*****
    BITAND
    *****/
else if (strcmp(token,"bitAnd") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = BITAND;
}
/*****
```



```
SHIFTL
*****/
else if (strcmp(token,"shiftL") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SHIFTL;
}
/*****
SHIFTR
*****/
else if (strcmp(token,"shiftR") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SHIFTR;
}
/*****
AND
*****/
else if (strcmp(token,"and") == 0)
{
    if (WhereAmI == PREMISE)
    {
        strcat (Comp, token);
        PrevChoice[ChoiceCounter++] = AND;
        Equation = LHS;
    }
}
/*****
NOT
*****/
else if (strcmp(token,"not") == 0)
{
    PrevChoice[ChoiceCounter++] = NOT;
    strcat (Comp, token);
}
/*****
SQRT
*****/
else if (strcmp(token,"sqrt") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SQRT;
}
/*****
POWER
*****/
else if (strcmp(token,"power") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = POWER;
}
/*****
EXP
*****/
else if (strcmp(token,"exp") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = EXP;
}
/*****
LOG
*****/
else if (strcmp(token,"log") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = LOG;
}
/*****
PI
*****/
else if ((strcmp(token,"p") == 0) || (strcmp(token, "PI") == 0))
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = PI;
}
/*****
COS
*****/
```

90/06/07
13:14:16

retrieve.c

8

```
else if (strcmp(token,"cos") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = COS;
}
/*****
    ACOS
*****/
else if (strcmp(token,"acos") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ACOS;
}
/*****
    SIN
*****/
else if (strcmp(token,"sin") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SIN;
}
/*****
    ASIN
*****/
else if (strcmp(token,"asin") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ASIN;
}
/*****
    TAN
*****/
else if (strcmp(token,"tan") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = TAN;
}
/*****
    ATAN
*****/
else if (strcmp(token,"atan") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ATAN;
}
/*****
    LT
*****/
else if (strcmp(token,"<") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = LT;
    Equation = RHS;
}
/*****
    GT
*****/
else if (strcmp(token,">") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = GT;
    Equation = RHS;
}
/*****
    ADD
*****/
else if (strcmp(token,"+") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = ADD;
}
/*****
    SUBTRACT
*****/
else if (strcmp(token,"-") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SUBTRACT;
```

90/06/07
13:14:16

retrieve.c

9

```

/*****
MULTIPLY
*****/
else if (strcmp(token,"*") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = MULTIPLY;
}
/*****
DIVIDE
*****/
else if (strcmp(token,"/") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = DIVIDE;
}
/*****
LE
*****/
else if (strcmp(token,"<=") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = LE;
    Equation = RHS;
}
/*****
GE
*****/
else if (strcmp(token,">=") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = GE;
    Equation = RHS;
}
/*****
NE
*****/
else if (strcmp(token,"8") == 0 || strcmp(token,"<>") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = NE;
    Equation = RHS;
}
/*****
EQ (in consequence)
*****/
else if (strcmp(token,"=") == 0 || strcmp(token,"==") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = EQ;
    Equation = RHS;
}
/*****
L_PAREN
*****/
else if (strcmp(token,"(") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = L_PAREN;
    if(FuncParenCount > 0)
        FuncParenCount++;
    ParenCount++;
}
/*****
R_PAREN
*****/
else if (strcmp(token,")") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = R_PAREN;
    if(FuncParenCount > 0)
        FuncParenCount--;
    ParenCount--;
}
/*****
COMMA
*****/

```

```
else if (strcmp(token, ",") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = COMMA;
}
/*****
    SET
*****/
else if (strcmp(token, "set") == 0)
{
    strcat (Comp, token);
    PrevChoice[ChoiceCounter++] = SET;
    Equation = LHS;
}
/*****
    NUMBERS and VARIABLES
*****/
else
{
    /*****
        Determine whether it was msid, signal, local and set
        the ChoiceCounter token history. If it's not defined
        locally we will assume it's a number.
        *****/
    var_check (token, &local_index, &global_index, "null");
    if (local_index >= 0)
    {
        strcat (Comp, token);
        if (strcmp (CompVars[local_index].class, "msid") == 0)
            PrevChoice[ChoiceCounter++] = MSID;
        else if (strcmp (CompVars[local_index].class, "signal") == 0)
            PrevChoice[ChoiceCounter++] = SIGNAL;
        else if (strcmp (CompVars[local_index].class, "local") == 0)
            PrevChoice[ChoiceCounter++] = LOCAL;
        /*****
            Set the type for this comparison if on the LHS
            *****/
        if (Equation == LHS)
            strcpy (CompareType[NumberOfCompares++], CompVars[local_index].type);
    }
    else
    {
        if (funcCheck (token) != ERROR)
        {
            strcat (Comp, token);
            PrevChoice[ChoiceCounter++] = FUNCTION;
        }
        else if (str_isalnum (token) != ERROR)
        {
            strcat (Comp, token);
            PrevChoice[ChoiceCounter++] = NUMBER;
        }
        else
        {
            /*****
                Flag the unknown token to the user, if
                it has not been flagged before.
                *****/
            if (strncmp (token, "...", 3) != 0)
            {
                strcat (Comp, "...");
                strcat (Comp, token);
                strcat (Comp, "...");
            }
            else strcat (Comp, token);
            CompInfo[CompNumber].disposition = ERROR;
        }
    }
}
} /* end of numbers or variables */
} /* end of while loop */
return (OK); /* We done good buck-a-roo */
/* end of read_tokens () */
```

90/06/07
13:14:19

save.c

1

SAVE

Purpose: Save is for saving the Comp and its necessary files

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 5/15/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

```
*****/

/*****<---->*****/
*
* MODULE NAME: save ( show_message )
*
*
* MODULE FUNCTION:
*
*   Very minor changes to the original NASA version of this file were made to modify
*   the cursor during the save operation.
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"

save( show_message )

    int show_message;

(
    int i; /* Counter */

    char explain[200], /* Text message repository */
        highLevelFile[PATH_LEN], /* Path for CODE lang. file */
        compVarFile[PATH_LEN], /* Path for comp variable file */
        groupDatFile[PATH_LEN]; /* Path for the [GroupName].dat file */

    FILE *filePtr, /* file pointers */
        *openFile(); /* A handy file opening routine */

    select_cursor( Clock_Cursor );

    /*
     * Since we're saving, that means the group has been modified
     * So we better change the groups disposition to COPMLETE and
     * remove the old executable if it had previously been INSTALLED.
     */
    )
```

```
if(GroupInfo[GroupNumber].disposition == INSTALLED)
{
    GroupInfo[GroupNumber].disposition = COMPLETE;
    sprintf(explain, "rm -f %s/%s 2>>/tmp/code.err", AMGroups, GroupName);
    system(explain);
}

/*
 * Let's rebuild the paths in case the comp name has changed since
 * our last save.
 */

sprintf (highLevelFile, "%s/%s/%s.h", CodeGroups, GroupName, CompName);
sprintf (compVarFile, "%s/%s/%s.v", CodeGroups, GroupName, CompName);
sprintf (groupDatFile, "%s/%s.dat", AMSupport, GroupName);

/*
 * Open CODE language file and write comp.
 */

if (!(filePtr = openFile (highLevelFile, "w", "save")))
    return (ERROR);

/*
 * Put the CODE language string into it's file
 */

fprintf (filePtr, "%s", Comp);
fclose (filePtr);

/*****
 * Open comp variable file and write variables
 *****/

if (!(filePtr = openFile (compVarFile, "w", "save")))
    return (ERROR);

/*****
 * Write all of the variables for the comp with header
 *****/
fprintf (filePtr, "#variable_name type class occurrences put_or_get lo1_limit lo2_limit hi1_limit hi2_limit nomenclature\n");
for (i=0; i<NumCompVars; i++)
{
    fprintf(filePtr, "%s %s %s %d %d %f %f %f %f %s\n",
        CompVars[i].name,
        CompVars[i].type,
        CompVars[i].class,
        CompVars[i].occurrence,
        CompVars[i].put_or_get,
        CompVars[i].lo1_limit,
        CompVars[i].lo2_limit,
        CompVars[i].hi1_limit,
        CompVars[i].hi2_limit,
        CompVars[i].nomenclature);
}
fclose (filePtr);

/*****
 * Open the [GroupName].dat file for writing
 *****/
if (!(filePtr = openFile (groupDatFile, "w", "save")))
    return (ERROR);

/*****
 * Write all of the group info for the group with header.
 *****/
fprintf (filePtr, "#name_name noise_filter rate on_off disposition\n");
for (i=0; i < NumOfComps; i++)
{
    fprintf (filePtr, "%s %d %d %d %d %d %s\n", CompInfo[i].name,
        CompInfo[i].noise_filter,
        CompInfo[i].rate,
        CompInfo[i].on_off,
        CompInfo[i].disposition,
        CompInfo[i].cycle_mode,
        CompInfo[i].purpose);
}
```

90/06/07
13:14:19

save.c

3

```
    }
    fclose (filePtr);

    /*
     * Put the group names into the GroupInfo file
     */
    if (writeGroupNames() == ERROR)
        return (ERROR);

    /*
     * Tell the user we have saved their comp.
     */

    select_cursor( Shuttle_Cursor );

    if ( show_message )
    {
        sprintf (explain,"%s has been saved", CompName);
        user_ack( explain, HELP_U_ACK );
    }
    NeedToSave = FALSE;
    return (OK);
} /* end of save */
```

90/06/07
13:14:21

strins.c

1

```
static char SccsId[] = "@(#)strins.c 2.1 8/17/88 08:55:36";
```

```
/*-----  
*  
* Function:      strins  
*  
* Entry specification:  
*   char *strins(into, from)  
*   register char *into, *from;  
*  
* Description:  
*   This routine inserts a string into another string, it is similar to the  
*   strcat found in the C library (except strcat inserts after).  
*  
* Inputs:  
*   into   Pointer-> target string  
*   from   Pointer-> string to insert  
*  
* Returns: Nothing  
*  
* External references:      None  
* Resources used:           None  
* Limitations:              None  
* Assumptions:              None  
*  
* Written by:  Tom Silva, Bruce G. Jackson & Associates  
*  
* Traceability:  
*   Version   Date       Description  
*   -----  
*   1.0       09/01/87    initial version  
*  
* Notes:      None  
*-----*/
```



```
/*-----*/
char *strins(into, from)
register char *into, *from;
{
    register char *to;
    register int to_size, from_size;

    /*-----
    * Find the size of both strings. Include the zero byte for the size of
    * the 'into' string (the zero byte must be moved with the string).
    * Leave the pointers pointing at the zero bytes of the strings.
    */
    for (from_size = 0; *from; from++, from_size++);

    for (to = into, to_size = 1; *to; to++, to_size++);

    /*-----
    * Slide the 'into' string forward by copying it from it's tail to it's
    * head. It needs to be moved forward by the number of bytes in the 'from'
    * string. Then copy the 'from' string into the open slot.
    */
    for (into = to, to += from_size; to_size > 0; to--, into--, to_size--)
        *to = *into;
    for (from--; from_size > 0; to--, from--, from_size--)
        *to = *from;
}
```

90/06/07
13:14:25

token_help.c

1

```
#include <stdio.h>
#include <sys/file.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include "code.h"
#include "widgets.h"
```

```
/*-----*/
*
* MODULE NAME: token_help( token, theMODE, widget )
*
*
* MODULE FUNCTION:
*
*   Displays the help text and performs the LIST functions.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
void token_help( token, theMODE, widget )
```

```
    int    theMODE,
           token;
    Widget widget;
```

```
{
    char    cmd    [ CMD_LEN ],
           listString[ 5000 ];
    int     i;
```

```
    if ( theMODE == HELP )
    {
```

```
        /*
         * Make sure the documentation file exists.
         */
```

```
        if ( access(CodeDocs,F_OK) == ERROR )
        {
            user_ack("Code documentation does not exist on this machine", HELP_U_ACK);
            return;
        }
```

```
        /*
         * Check token structure to ensure help text exists for the current token.
         */
```

```
        if ( tokens[ token ].help == HELP_TEXT_NO )
        {
            user_ack("You have chosen an undocumented token, please notify the developers", HELP_U_ACK);
            return;
        }
```

```
        /*
         * Build a temporary file containing the help text for the current token.
         */
```

```
        select_cursor( Clock_Cursor );
```

90/06/07
13:14:25

token_help.c

2

```
strcpy( cmd, "more +/\042*" );
strcat( cmd, tokens[ token ].name );
strcat( cmd, "\042 " );
strcat( cmd, CodeDocs );
strcat( cmd, "> /tmp/code.tmp 2>/tmp/code.err" );
if( system( cmd ) != OK )
{
    select_cursor( Shuttle_Cursor );
    user_ack("Error reading documentation file, help text is not available", HELP_U_ACK);
    return;
}

select_cursor( Shuttle_Cursor );

/*
 * Clear the help text buffer, assign the temporary disk file to the text
 * widget, move the widget to the middle of the screen, show the help text
 * popup.
 */

display_file( HELP, "/tmp/code.tmp" );

/*
 * Delete the temporary file.
 */

if ( system("rm /tmp/code.tmp >>/tmp/code.err 2>&1") != OK )
    user_ack("Unable to delete: /tmp/code.tmp - help text temporary file", HELP_U_ACK);
}

/*
 * The user wishes to list either: MSIDs, signals, or user functions.
 */

else if ( theMODE == LIST )
    switch (token)
    {
        case MSID:      if ( access(MSIDTbl,F_OK) == ERROR )
                        user_ack( "MSID table not found", HELP_U_ACK );
                        else
                            display_file( LIST, MSIDTbl );
                        break;

        case SIGNAL:    if ( access(SignalTbl,F_OK) == ERROR )
                        user_ack( "SIGNAL table not found", HELP_U_ACK );
                        else
                            display_file( LIST, SignalTbl );
                        break;

        case FUNCTION:  select_cursor( Clock_Cursor );
                        strcpy (listString, "\n\tListing of User Functions");
                        strcat (listString, "\n\t_____\n\n");
                        for (i=0;i<NumberOfUserFuncs;i++)
                        {
                            strcat( listString, "\t" );
                            strcat( listString, UserFuncs[i]);
                            strcat( listString, "\n");
                        }
                        select_cursor( Shuttle_Cursor );
                        display_str( LIST, listString );
                        break;

        default:        user_ack("A listing is not available for this choice", HELP_U_ACK);
    }
}
```

90/06/07
13:14:28

tokens.h

1

```
*****<----->*****
*
* FILE NAME:    tokens.h
*
*
* FILE FUNCTION:
*
* This file contains the tokens[] array which defines the following information for
* each CODE push button:
*   1) availability of help text
*   2) help text locator
*   3) comp token input function
*   4) comp token input function argument
*
* NOTE: do not change the order of the structure elements, as they match the
*       constants defined in: code_const.h
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   N/A
*
*****<----->*****/
```

```
struct token_tag tokens[] = {

    { "TOC",           HELP_TEXT_YES, NULL,          NULL },
    { "CREATE",        HELP_TEXT_YES, NULL,          NULL },
    { "EDIT",          HELP_TEXT_YES, NULL,          NULL },
    { "DELETE",        HELP_TEXT_YES, NULL,          NULL },
    { "IF",            HELP_TEXT_YES, (*put_if),      NULL },
    { "ELSE",          HELP_TEXT_YES, (*put_else),    NULL },
    { "AND",           HELP_TEXT_YES, (*put_and),     NULL },
    { "BITx",          HELP_TEXT_NO,  (*put_bitXor),  NULL },
    { "BITa",          HELP_TEXT_NO,  (*put_bitAnd),  NULL },
    { "THEN",          HELP_TEXT_YES, (*put_then),    NULL },
    { "ENDIF",         HELP_TEXT_YES, (*put_endif),   NULL },
    { "OR",            HELP_TEXT_YES, (*put_or),      NULL },
    { "NOT",           HELP_TEXT_YES, (*put_not),     NULL },
    { "BITo",          HELP_TEXT_NO,  (*put_bitOr),   NULL },
    { "MSID",          HELP_TEXT_YES, (*put_msid),    NULL },
    { "LOCAL",         HELP_TEXT_YES, (*put_local),   NULL },
    { "STRING",        HELP_TEXT_NO,  (*put_string),  NULL },
    { "SIGNAL",        HELP_TEXT_YES, (*put_signal),  NULL },
    { "NUMBER",        HELP_TEXT_YES, (*put_number),  NULL },
    { "SHORT",         HELP_TEXT_NO,  NULL,          NULL },
    { "FLOAT",         HELP_TEXT_YES, NULL,          NULL },
    { "CHAR",          HELP_TEXT_NO,  NULL,          NULL },
    { "INTEGER",       HELP_TEXT_YES, NULL,          NULL },
    { "DOUBLE",        HELP_TEXT_YES, NULL,          NULL },
    { "EQ",            HELP_TEXT_NO,  (*put_eq),      NULL },
    { "NE",            HELP_TEXT_NO,  (*put_ne),      NULL },
    { "LE",            HELP_TEXT_NO,  (*put_le),      NULL },
    { "GE",            HELP_TEXT_NO,  (*put_ge),      NULL },
    { "LT",            HELP_TEXT_NO,  (*put_lt),      NULL },
    { "GT",            HELP_TEXT_NO,  (*put_gt),      NULL },
    { "SET",           HELP_TEXT_YES, (*put_set),     NULL },
    { "PRINT",         HELP_TEXT_YES, (*put_print),   NULL },
    { "FUNCTION",      HELP_TEXT_YES, (*put_func),    FUNCTION },
    { "COMMENT",       HELP_TEXT_YES, (*put_comment), NULL },
    { "START",         HELP_TEXT_NO,  (*put_start),   NULL },
    { "STOP",          HELP_TEXT_NO,  (*put_stop),    NULL },
    { "RETRIEVE",      HELP_TEXT_YES, NULL,          NULL },
    { "SAVE",          HELP_TEXT_YES, NULL,          NULL },
    { "INSTALL",       HELP_TEXT_YES, NULL,          NULL },
    { "COPY",          HELP_TEXT_NO,  NULL,          NULL },
    { "REMOVE",        HELP_TEXT_YES, NULL,          NULL },
    { "HARDCOPY",      HELP_TEXT_YES, NULL,          NULL },
    { "BACKUP",        HELP_TEXT_YES, NULL,          NULL },
    { "QUIT",          HELP_TEXT_YES, NULL,          NULL },
    { "HELP",          HELP_TEXT_YES, NULL,          NULL },
}
```

tokens.h

```
{ "LIST",      HELP_TEXT_YES, NULL,      NULL },
{ "+",         HELP_TEXT_NO,  (*put_add),  NULL },
{ "-",         HELP_TEXT_NO,  (*put_subtract), NULL },
{ "*",         HELP_TEXT_NO,  (*put_multiply), NULL },
{ "/",         HELP_TEXT_NO,  (*put_divide),  NULL },
{ "(",         HELP_TEXT_NO,  (*put_l_paren),  NULL },
{ ")",         HELP_TEXT_NO,  (*put_r_paren),  NULL },
{ ",",         HELP_TEXT_NO,  (*put_comma),    NULL },
{ "PI",        HELP_TEXT_NO,  (*put_pi),      NULL },
{ "SQRT",      HELP_TEXT_NO,  (*put_func),    SQRT },
{ "POWER",     HELP_TEXT_NO,  (*put_func),    POWER },
{ "EXP",       HELP_TEXT_NO,  (*put_func),    EXP },
{ "LOG",       HELP_TEXT_NO,  (*put_func),    LOG },
{ "shiftL",   HELP_TEXT_NO,  (*put_shiftL), NULL },
{ "shiftR",   HELP_TEXT_NO,  (*put_shiftR), NULL },
{ "COS",       HELP_TEXT_NO,  (*put_func),    COS },
{ "SIN",       HELP_TEXT_NO,  (*put_func),    SIN },
{ "TAN",       HELP_TEXT_NO,  (*put_func),    TAN },
{ "ACOS",     HELP_TEXT_NO,  (*put_func),    ACOS },
{ "ASIN",     HELP_TEXT_NO,  (*put_func),    ASIN },
{ "ATAN",     HELP_TEXT_NO,  (*put_func),    ATAN }, /* #65 */
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ NULLS,      HELP_TEXT_NO,  NULL,          NULL },
{ "Group-Comp", HELP_TEXT_YES, NULL,      NULL }, /* #80 */
{ "Group-Sel",  HELP_TEXT_YES, NULL,      NULL },
{ "Comp-Sel",   HELP_TEXT_YES, NULL,      NULL },
{ "Set-Fonts",  HELP_TEXT_YES, NULL,      NULL },
{ "User-Ack",   HELP_TEXT_YES, NULL,      NULL },
{ "Help-Help",  HELP_TEXT_YES, NULL,      NULL },
{ "Help-Device", HELP_TEXT_YES, NULL,      NULL },
```

};

1

```

int compIndex, /* Index to the CODE lang. comp string */
tokenIndex, /* Index to an individual token */
clangIndex, /* Index to the C lang. comp */
spaceIndex, /* Index to spacesEtAl */
equation, /* Similar to global equation in code.h */
numberOfIfs, /* The number of if we have parsed */
numberOfEndifs, /* The number of endifs we have parsed */
numberOfPrints, /* The number of print we have parsed */
numberOfSets, /* The number of sets we have parsed */
lastToken, /* Indicates the previous token type */
i, /* Simple counters */
fltColor, /* Used for parsing the fault color number out of the msg */
givenHead, /* TRUE after we have processed the comp header */
varIndex, /* The index value set in the function getVarIndex */
relIndex, /* The relative index of an MSID used in getVarIndex */
varClass, /* The return value from the function getVarIndex */
charIndex, /* The index of the characters for a variable in a print str */
whereAmI, /* The local version of the one defined in code.h */
numToken, /* TRUE if the first token has been found */
sigShortArraySize, /* The size of the array of structures for noise filter output */
sigIntArraySize, /* The size of the array of structures for noise filter output */
sigFloatArraySize, /* The size of the array of structures for noise filter output */
sigDoubleArraySize, /* The size of the array of structures for noise filter output */
sigUnsignedArraySize, /* The size of the array of structures for noise filter output */
sigLongArraySize, /* The size of the array of structures for noise filter output */
sigOffShortArraySize, /* The size of the array of structures for noise filter output */
sigOffIntArraySize, /* The size of the array of structures for noise filter output */
sigOffFloatArraySize, /* The size of the array of structures for noise filter output */

```

90/06/07
13:14:31

translate.c

2

```
sigOffDoubleArraySize, /* The size of the array of structures for noise filter output */
sigOffUnsignedArraySize, /* The size of the array of structures for noise filter output */
sigOffLongArraySize, /* The size of the array of structures for noise filter output */
sigOffStringArraySize, /* The size of the array of structures for noise filter output */
sigStringArraySize; /* The size of the array of structures for noise filter output */

char clangVersion[MAX_COMP_LEN], /* A string to hold the comp's C language version */
token[500], /* A string to hold a token */
compName_c[PATH_LEN], /* Path for C language file */
comp[MAX_COMP_LEN], /* The all important string for holding the comp string */
tmp1[169], /* What would life be without a few transient strings */
tmp2[169], /* Dito my older brother */
msgString[169], /* What would life be without a few transient strings */
varString[MAX_VAR_LEN], /* The string for pulling vars from prints */
lastVarType[TYPE_LEN], /* The type of the last found variable */
strOperator[4], /* The operator used with a string, usually == or != */
compHeader[1000], /* Temporary storage for the comp's header */
endString[250], /* Used to store variable names in formatted prints */
amSprintfing; /* Flag used to know if inside a sprintf statement */

/*
 * Read the comp into the string comp
 */
if (readCODEFile(compName, comp) == ERROR)
{
    sprintf(msgString, "Install: There was a problem reading the CODE file for %s", compName);
    user_ack(msgString, HELP_U_ACK);
    return(ERROR);
}

/*****
 * Initialize some soon-to-be-used variables
 *****/
sigShortArraySize = sigIntArraySize = sigFloatArraySize = sigLongArraySize = 0;
sigDoubleArraySize = sigUnsignedArraySize = sigStringArraySize = 0;
sigOffShortArraySize = sigOffIntArraySize = sigOffFloatArraySize = 0;
sigOffDoubleArraySize = sigOffUnsignedArraySize = sigOffStringArraySize = 0;
numberOfIfs = compIndex = clangIndex = givenHead = sigOffLongArraySize = 0;
numToken = numberOfEndifs = numberOfSets = numberOfPrints = spaceIndex = 0;
clangVersion[0] = compHeader[0] = 0;
amSprintfing = FALSE;

/*****
 * Loop through the CODE language string, translating
 * tokens to C, until the end of the string is reached.
 *****/
do
{
    /*****
     * Yank out the control characters and spaces,
     * we don't care about them spaces and such.
     *****/
    yankBlank (comp, &compIndex);

    /*****
     * YANK A TOKEN out of the Comp's string. Tokens
     * are delimited by spaces, tabs, or line feeds.
     *****/
    yankToken (comp, &compIndex, token, &tokenIndex);

    /*****
     * Now the real fun starts. Match the token we
     * just yanked with the allowable set of CODE
     * tokens and translate appropriately.
     *****/
    COMMENTS
    *****/
    if (strncmp(token, "/*", 2) == 0)
    {
        /*****
         * If this comment is not complete we need to get the rest.
         *****/
        if (token[strlen(token)-1] != '/')
        {
            token[tokenIndex] = comp[compIndex];
            while (!{comp[compIndex-1] == '*' && comp[compIndex] == '/'})
                token[tokenIndex++] = comp[compIndex++];
            token[tokenIndex++] = comp[compIndex++];
        }
    }
}
```

```
token[tokenIndex] = 0;
}
if (!(givenHead))
{
    sprintf (compHeader, "%s\n%s()\n", token, compName);
    givenHead = TRUE;
}
/*****
Else add the comment token to the end of the C lang. string.
*****/
else strcat (clangVersion, token);

lastToken = COMMENT;
}
/*****
IF
*****/
else if (strcmp(token,"if") == 0)
{
    numberOfIfs++;
    /*****
    Cat a ';' to close off the last statement
    *****/
    if (lastToken == MSID || lastToken == R_PAREN)
        strcat (clangVersion, ";");
    strcat (clangVersion, "\n");
    for (i=0;i<(numberOfIfs - numberOfEndifs);i++)
        strcat (clangVersion, "\t");
    /*****
    Put in a second left paren to surround the
    comparitors and ensure proper precedence
    *****/
    strcat (clangVersion, "if ( (");
    lastToken = IF;
    whereAmI = PREMISE;
    equation = LHS;
} /* end of IF */
/*****
ENDIF
*****/
else if (strcmp(token,"endif") == 0)
{
    /*****
    Cat a ';' to close off the last statement
    *****/
    strcat (clangVersion, ";\n");
    for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
        strcat (clangVersion, "\t");
    strcat (clangVersion, ")\n");
    lastToken = END_IF;
    numberOfEndifs++;
}
/*****
THEN
*****/
else if (strcmp(token,"then") == 0)
{
    /*****
    Figure out and set the indentation
    *****/
    strcat (clangVersion, " )\n");
    for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
        strcat (clangVersion, "\t");
    strcat (clangVersion, "{");
    equation = LHS;
    whereAmI = CONSEQUENCE;
    lastToken = THEN;
}
/*****
ELSE
*****/
else if (strcmp(token,"else") == 0)
{
    /*****
    Figure out and set the indentation
    *****/
    strcat (clangVersion, ";\n");
    for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
```



```
        strcat (clangVersion, "\t");
        strcat (clangVersion, "}\n");
        for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
            strcat (clangVersion, "\t");
        strcat (clangVersion, "else\n");
        for (i=0;i<(numberOfIfs-numberOfEndifs);i++)
            strcat (clangVersion, "\t");
        strcat (clangVersion, "{");
        equation = LHS;
        lastToken = ELSE;
    }
/*****
    OR
*****/
else if (strcmp(token,"or") == 0)
{
    strcat (clangVersion, " ) || (");
    equation = LHS;
    lastToken = OR;
}
/*****
    POWER
*****/
else if (strcmp(token,"power") == 0)
{
    strcat (clangVersion, " pow");
    lastToken = POWER;
}
/*****
    EXP
*****/
else if (strcmp(token,"exp") == 0)
{
    strcat (clangVersion, " exp");
    lastToken = EXP;
}
/*****
    BITOR
*****/
else if (strcmp(token,"bitOr") == 0)
{
    strcat (clangVersion, " |");
    lastToken = BITOR;
}
/*****
    BITAND
*****/
else if (strcmp(token,"bitAnd") == 0)
{
    equation = LHS;
    strcat (clangVersion, " &");
    lastToken = BITAND;
}
/*****
    BITXOR
*****/
else if (strcmp(token,"xor") == 0 || strcmp(token, "bitXor") == 0)
{
    equation = LHS;
    strcat (clangVersion, " ^");
    lastToken = BITXOR;
}
/*****
    AND
*****/
else if (strcmp(token,"and") == 0)
{
    equation = LHS;
/*****
    The following is conditional to support
    an older syntax which had 'and's in the
    CONSEQUENCE
*****/
    if (whereAmI == PREMISE)
        strcat (clangVersion, " ) && (");
    lastToken = AND;
}
/*****
```

```
NOT
*****/
else if (strcmp(token,"not") == 0)
{
    strcat (clangVersion, "!");
    lastToken = NOT;
}
/*****
NOT EQUAL TO
*****/
else if (strcmp(token,"<>") == 0 || strcmp(token,"8") == 0)
{
    if (strcmp (lastVarType, "char") != 0)
        strcat (clangVersion, " !=");
    else strcpy (strOperator, " !=");
    equation = RHS;
    lastToken = NE;
}
/*****
ADD
*****/
else if (strcmp(token,"+") == 0)
{
    if (strcmp (lastVarType, "char") == 0 || varClass == STRING)
        strcat (clangVersion, ",");
    else strcat (clangVersion, " +");
    lastToken = ADD;
}
/*****
SUBTRACT
*****/
else if (strcmp(token,"-") == 0)
{
    strcat (clangVersion, " -");
    lastToken = SUBTRACT;
}
/*****
MULTIPLY
*****/
else if (strcmp(token,"*") == 0)
{
    strcat (clangVersion, " *");
    lastToken = MULTIPLY;
}
/*****
DIVIDE
*****/
else if (strcmp(token,"/") == 0)
{
    strcat (clangVersion, " /");
    lastToken = DIVIDE;
}
/*****
P
*****/
else if ((strcmp(token,"p") == 0) || (strcmp(token,"PI") == 0))
{
    strcat (clangVersion, " M_PI");
    lastToken = PI;
}
/*****
EQEQ
*****/
else if (strcmp(token,"==") == 0)
{
    if (strcmp (lastVarType, "char") != 0)
        strcat (clangVersion, " ==");
    else strcpy (strOperator, " ==");
    equation = RHS;
    lastToken = EQ;
}
/*****
EQ
*****/
else if (strcmp(token,"=") == 0)
{
    if (whereAmI == PREMISE)
        strcat (clangVersion, " ==");
```

90/06/07
13:14:31

translate.c

6

```
    else if (strcmp (lastVarType, "char") != 0)
        strcat (clangVersion, " =");
    else strcpy (strOperator, " =");
    equation = RHS;
    lastToken = EQ;
}
/*****
    GT
*****/
else if (strcmp(token, ">") == 0)
{
    if (strcmp (lastVarType, "char") != 0)
        strcat (clangVersion, " >");
    else strcpy (strOperator, " >");
    equation = RHS;
    lastToken = GT;
}
/*****
    GE
*****/
else if (strcmp(token, ">=") == 0)
{
    if (strcmp (lastVarType, "char") != 0)
        strcat (clangVersion, " >=");
    else strcpy (strOperator, " >=");
    equation = RHS;
    lastToken = GE;
}
/*****
    LT
*****/
else if (strcmp(token, "<") == 0)
{
    if (strcmp (lastVarType, "char") != 0)
        strcat (clangVersion, " <");
    else strcpy (strOperator, " <");
    equation = RHS;
    lastToken = LT;
}
/*****
    LE
*****/
else if (strcmp(token, "<=") == 0)
{
    if (strcmp (lastVarType, "char") != 0)
        strcat (clangVersion, " <=");
    else strcpy (strOperator, " <=");
    equation = RHS;
    lastToken = LE;
}
/*****
    SHIFTL
*****/
else if (strcmp(token, "shiftL") == 0)
{
    strcat (clangVersion, " <<");
    lastToken = SHIFTL;
}
/*****
    SHIFTR
*****/
else if (strcmp(token, "shiftR") == 0)
{
    strcat (clangVersion, " >>");
    lastToken = SHIFTR;
}
/*****
    LEFT PAREN
*****/
else if (strcmp(token, "(") == 0)
{
    strcat (clangVersion, " (");
    lastToken = L_PAREN;
}
/*****
    RIGHT PAREN
*****/
else if (strcmp(token, ")") == 0)
```

```
(
    strcat (clangVersion, " ");
    lastToken = R_PAREN;
)
/*****
    SET
*****/
else if (strcmp(token,"set") == 0)
{
    if(!(lastToken == THEN || lastToken == ELSE || lastToken == END_IF) &&
        (numberOfSets > 0 || numberOfPrints > 0))
        strcat(clangVersion, ";");
    lastToken = SET;
    whereAmI = CONSEQUENCE;
    equation = LHS;
    numberOfSets++;
}
/*****
    PRINT
*****/
else if (strncmp(token, "print", 5) == 0)
{
    /*****
        Put a semi-colon after the last line if need be
        *****/
    if(!(lastToken == THEN || lastToken == ELSE || lastToken == END_IF) &&
        (numberOfSets > 0 || numberOfPrints > 0))
        strcat(clangVersion, ";");
    /*****
        Pull the fault class number
        *****/
    tmp1[0]=token[5];
    tmp1[1]=0;
    fltColor = atoi(tmp1);
    compIndex++; /* Dispose of a space */
    tokenIndex = 0; /* Start again for the message */
    endString[0] = 0;
    varString[0] = 0;
    /*****
        Get the first double quote then loop
        until we find the second.
        *****/
    token[tokenIndex++] = comp[compIndex++];
    do
    {
        token[tokenIndex++] = comp[compIndex++];
        /*****
            Break off here to process variables. Yes its true that
            variables should be preceded by a '%' to be recognized here.
            *****/
        if (comp[compIndex-1] == '%')
        {
            /*****
                Get the whole variable
                *****/
            charIndex = 0;
            while ((comp[compIndex] >= 'a' && comp[compIndex] <= 'z') ||
                (comp[compIndex] >= 'A' && comp[compIndex] <= 'Z') ||
                (comp[compIndex] >= '0' && comp[compIndex] <= '9') ||
                comp[compIndex] == '_')
                varString[charIndex++] = comp[compIndex++];
            varString[charIndex] = 0;
            upper (varString); /* Convert to upper case */
            /*****
                Use getVarIndex to find out the class and index
                *****/
            if ((varClass = getVarIndex(varString,CompVars,
                numCompVars,&varIndex,&relIndex)) != ERROR)
            {
                /*****
                    Determine the var. type
                    *****/
                if (CompVars[varIndex].type[0] == 's')
                    token[tokenIndex++] = 'h';
                else if (CompVars[varIndex].type[0] == 'c')
                    token[tokenIndex++] = 's';
                else if (CompVars[varIndex].type[0] == 'f')
                    token[tokenIndex++] = 'f';
            }
        }
    }
}
```

translate.c

```

else if (CompVars[varIndex].type[0] == 'i')
    token[tokenIndex++] = 'd';
else if (CompVars[varIndex].type[0] == 'd')
{
    token[tokenIndex++] = 'l';
    token[tokenIndex++] = 'f';
}
else /* Don't know this type */
{
    sprintf(msgString, "Translate: '%s' has unknown type '%s'; aborting translation..."
, varString, CompVars[varIndex].type);
    user_ack(msgString, HELP_U_ACK);
    return (ERROR);
}
/*****
MSIDs require an array call with index
*****/
if (varClass == MSID)
{
    sprintf(tmp1, "value[%d]", relIndex);
    strcat(endString, tmp1);
    /*****
    Add a comma expecting another variable
    *****/
    strcat(endString, ",");
}
/*****
SIGNALS & locals can just be appended
*****/
else if (varClass == SIGNAL || varClass == LOCAL)
{
    strcat(endString, varString);
    /*****
    Add a comma expecting another variable
    *****/
    strcat(endString, ",");
}
else
{
    sprintf(msgString, "%s is an unknown variable to comp %s.", varString, compName);
    user_ack(msgString, HELP_U_ACK);
    strcat(clangVersion, varString);
}
} /* end of variables */
} while (comp[compIndex-1] != '\0');
token[tokenIndex] = 0;
/*****
Remove the last comma from the endString
*****/
endString[strlen(endString)-1] = 0;
/*****
Let's do some pretty formatting
*****/
strcat(clangVersion, "\n");
for (i=0; i<(numberOfIfs - numberOfEndifs)+1; i++)
    strcat(clangVersion, "\t");
/*****
sprintf the faultString
*****/
if (endString[0] == 0)
    sprintf(tmp1, "sprintf(faultString, %s);\n", token);
else /* we got %'s in the print string to add */
    sprintf(tmp1, "sprintf(faultString, %s, %s);\n", token, endString);
strcat(clangVersion, tmp1);
/*****
Let's do some pretty formatting again
*****/
for (i=0; i<(numberOfIfs - numberOfEndifs)+1; i++)
    strcat(clangVersion, "\t");
/*****
Now issue the fault message
*****/
sprintf(tmp1, "fltmsg_issue_NF (faultString, %d, %d, nf[nf_index], faultStruct)", fltColor, numberOfPr
ints);
strcat(clangVersion, tmp1);
/*****
Increment the number of fault messages for this comp

```

```

*****
numberOfPrints++;
/*****
Set the last token to PRINT
*****/
lastToken = PRINT;
}
/*****
VARIABLES
Right away we make a call to getVarIndex to determine wheter
the token is a variable, number of char string. If it is
we proceed, otherwise goodbye.
*****/
else if ((varClass=getVarIndex(token,CompVars,numCompVars,&varIndex,&relIndex)) !=ERROR)
{
    tmp1[0] = 0;tmp2[0] = 0; /* empty tmp1,tmp2 */
    /*
    * If the previous token was a set, then let's indent
    */
    if (lastToken == SET)
    {
        /*
        * If the type is char then we will be
        * sprintf'ing, otherwise not.
        */
        if (CompVars[varIndex].type[0] == 'c' || varClass == STRING)
            amSprintfing = TRUE;
        else amSprintfing = FALSE;

        /*****
        If we are setting a SIGNAL, then increment
        the size of the array of structures for
        signals of this type.
        *****/
        if (varClass == SIGNAL)
        {
            if (CompVars[varIndex].type[0] == 's') /* short */
                sigShortArraySize++;
            else if (CompVars[varIndex].type[0] == 'i') /* int */
                sigIntArraySize++;
            else if (CompVars[varIndex].type[0] == 'f') /* float */
                sigFloatArraySize++;
            else if (CompVars[varIndex].type[0] == 'd') /* double */
                sigDoubleArraySize++;
            else if (CompVars[varIndex].type[0] == 'l') /* long */
                sigLongArraySize++;
            else if (CompVars[varIndex].type[0] == 'u') /* unsigned */
                sigUnsignedArraySize++;
            else if (CompVars[varIndex].type[0] == 'c') /* string */
                sigStringArraySize++;
            else
            {
                sprintf (msgString, "Translate: %s was of unknown type '%s'...Aborting translation",token
, CompVars[varIndex].type);
                user_ack(msgString, HELP_U_ACK);
                return (ERROR);
            }
        }
        strcat (clangVersion, "\n");
        for (i=0;i<(numberOfIfs - numberOfEndifs)+1;i++)
            strcat (clangVersion, "\t");
    }
    else strcat (clangVersion, " ");

    /*
    * STRINGS (embedded variables handled with %)
    */
    if (varClass == STRING && token[strlen(token)-1] != '"')
    {
        do /* Grab chars until the double quote */
        {
            token[tokenIndex++] = comp[compIndex++];
            /*****
            Break off here to process variables. Yes its true that
            variables should be preceded by a '%' to be recognized here.
            *****/
            if (comp[compIndex-1] == '%')
            {

```

```

/*****
Get the whole variable
*****/
charIndex = 0;
while((comp[compIndex] >='a' && comp[compIndex] <='z') ||
      (comp[compIndex] >='A' && comp[compIndex] <='Z') ||
      (comp[compIndex] >='0' && comp[compIndex] <='9')) ||
      comp[compIndex] == '_'
    varString[charIndex++] = comp[compIndex++];
varString[charIndex] = 0;
upper (varString); /* Convert to upper case */
/*****
Use getVarIndex to find out the class and index
*****/
if ((varClass = getVarIndex(varString, CompVars,
                             numCompVars, &varIndex, &relIndex)) != ERROR)
{
    /*****
    Determine the var. type
    *****/
    if (CompVars[varIndex].type[0] == 's')
        token[tokenIndex++] = 'd';
    else if (CompVars[varIndex].type[0] == 'c')
        token[tokenIndex++] = 's';
    else if (CompVars[varIndex].type[0] == 'f')
        token[tokenIndex++] = 'f';
    else if (CompVars[varIndex].type[0] == 'i')
        token[tokenIndex++] = 'd';
    else if (CompVars[varIndex].type[0] == 'd')
    {
        token[tokenIndex++] = 'l';
        token[tokenIndex++] = 'f';
    }
    else /* Don't know this type */
    {
        sprintf (msgString, "Translate: '%s' has unknown type '%s'; aborting translation
...", varString, CompVars[varIndex].type);
        user_ack(msgString, HELP_U_ACK);
        return (ERROR);
    }
    /*****
    MSIDs require an array call with index
    *****/
    if (varClass == MSID)
    {
        /*
        * comma delimits
        */
        sprintf(tmp1, " value[%d],", relIndex);
        strcat (endString, tmp1);
    }
    /*****
    SIGNALS & locals can just be appended
    *****/
    else if (varClass == SIGNAL || varClass == LOCAL)
    {
        strcat (endString, varString);
        /*
        * Comma delimits
        */
        strcat (endString, ",");
    }
    else
    {
        sprintf (msgString, "%s is an unknown variable to comp %s.", varString, compName);
        user_ack(msgString, HELP_U_ACK);
        strcat (clangVersion, varString);
    }
}
} while (comp[compIndex-1] != ''); /* STRINGS, char variables */
token[tokenIndex] = 0;

/*
* Remove the last comma from the endString
*/
endString[strlen(endString)-1] = 0;

```

```
/*
 * sprintf the faultString
 */
if (endString[0] == 0)
    sprintf(tmp1,"%s);\n", token);
else /* we got %'s in the print string to add */
    sprintf(tmp1,"%s,%s);\n", token, endString);
strcat (clangVersion, tmp1);

/*
 * Set the last token to STRING
 */
lastToken = STRING;
amSprintfing = FALSE; /* re-init the bugger */
} /* STRINGS */
else /* STRINGS with embed space and all other variables */
{
    /*
     * Set the lastVarType in order to flag comparitors (e.g. EQEQ)
     */
    if (varClass == NUMBER)
        strcpy (lastVarType, "int");
    else strcpy (lastVarType, CompVars[varIndex].type);

    /*
     * If the variable is not a char string then cat it.
     */
    if(CompVars[varIndex].type[0]!='c' && varClass!=STRING && !amSprintfing)
    {
        if (varClass == MSID)
            sprintf (tmp1," value[%d]", relIndex);
        else sprintf (tmp1, "%s", token);
    }
    else /* It must be of type char or STRING w/embed space(s) */
    {
        if (whereAmI == PREMISE)
        {
            if (equation == LHS)
            {
                if (varClass == MSID)
                    sprintf (tmp1, "strcmp(value[%d],", relIndex);
                else /* SIGNAL or LOCAL */
                    sprintf (tmp1, "strcmp (%s,", token);
            }
            else /* RHS */
            {
                if (varClass == MSID)
                    sprintf(tmp1,"value[%d])%s 0",relIndex,strOperator);
                else /* SIGNAL, LOCAL */
                    sprintf (tmp1, "%s)%s 0", token, strOperator);
            }
        }
        else /* If CONSEQUENCE it must be a strcpy */
        {
            if (equation == LHS)
            {
                if(varClass == MSID)
                    sprintf (tmp1, "sprintf(value[%d],", relIndex);
                else sprintf (tmp1, "sprintf (%s,", token);
            }
            else /* RHS */
            {
                amSprintfing = FALSE; /* re-init the bugger */
                if(CompVars[varIndex].type[0]=='c' || varClass==STRING)
                    strcpy(tmp1,"%042%s%042");
                else if(CompVars[varIndex].type[0] == 'i' ||
                    CompVars[varIndex].type[0] == 's' ||
                    varClass == NUMBER)
                    strcpy(tmp1,"%042%d%042");
                else if(CompVars[varIndex].type[0] == 'd')
                    strcpy(tmp1,"%042%lf%042");
                else sprintf(tmp1,"%042%%c%042",CompVars[varIndex].type[0]);
                if (varClass == MSID)
                    sprintf(tmp2,"value[%d]",relIndex);
                else
                    sprintf (tmp2, "%s", token);
            }
            strcat (tmp1,tmp2);
        }
    }
}
```



```
    }
    strcat (clangVersion, tmp1);
    tmp1[0] = 0; tmp2[0] = 0; /* empty tmp1, tmp2 */
    lastToken = MSID; /* MSID ~ SIGNAL ~ LOCAL ~ NUMBER in this case */
}
/* end of variable */
/*****
UNRECOGNIZED TOKENS
If we have not recognized the token at this point we just drop
it in as is without any processing.
*****/
else
(
    strcat (clangVersion, " /* Unkown token */ ");
    strcat (clangVersion, token);
)
numToken++;
}while (comp[compIndex] != 0); /* Until we reach the end of the string */

if (lastToken != END_IF)
    strcat (clangVersion, ";");
/*****
Now that we have completed our token level translation
let's take care of putting the signals back into shared
memory where its necessary.
*****/
for (i=0; i<numCompVars; i++)
(
    if ((strcmp(CompVars[i].class, "signal")==0) &&
        (CompVars[i].put_or_get == PUT || CompVars[i].put_or_get == PET))
    (
        /*****
        Let's take care of indentation now!
        *****/
        strcat (clangVersion, "\n\t");

        sprintf (tmp2, "\\042%s\\042, %s, %d, nf[nf_index]);", CompVars[i].name, CompVars[i].name);
        switch (CompVars[i].type[0])
        {
            case 'c': sprintf(tmp1, "putsig_str_NF (\\042%s\\042, %s, %d, nf[nf_index], sigString);",
                               CompVars[i].name, CompVars[i].name, sigStringArraySize++);
                       break;

            case 's': sprintf(tmp1, "putsig_s_NF (\\042%s\\042, %s, %d, nf[nf_index], sigShort);",
                               CompVars[i].name, CompVars[i].name, sigShortArraySize++);
                       break;

            case 'i': sprintf(tmp1, "putsig_i_NF (\\042%s\\042, %s, %d, nf[nf_index], sigInt);",
                               CompVars[i].name, CompVars[i].name, sigIntArraySize++);
                       break;

            case 'f': sprintf(tmp1, "putsig_f_NF (\\042%s\\042, %s, %d, nf[nf_index], sigFloat);",
                               CompVars[i].name, CompVars[i].name, sigFloatArraySize++);
                       break;

            case 'd': sprintf(tmp1, "putsig_d_NF (\\042%s\\042, %s, %d, nf[nf_index], sigDouble);",
                               CompVars[i].name, CompVars[i].name, sigDoubleArraySize++);
                       break;

            case 'l': sprintf(tmp1, "putsig_l_NF (\\042%s\\042, %s, %d, nf[nf_index], sigLong);",
                               CompVars[i].name, CompVars[i].name, sigLongArraySize++);
                       break;

            case 'u': sprintf(tmp1, "putsig_u_NF (\\042%s\\042, %s, %d, nf[nf_index], sigUnsigned);",
                               CompVars[i].name, CompVars[i].name, sigUnsignedArraySize++);
                       break;

            default : break;
        }
        strcat (clangVersion, tmp1);
    )
}
/*****
Let's clean up by adding a few correctly indented '}'
*****/
if ((numberOfIfs - numberOfEndifs) > 0)
(
    strcat (clangVersion, "\n");

```

```
    for (i=0;i<(numberOfIfs - numberOfEndifs);i++)
        strcat (clangVersion, "\t");
    strcat (clangVersion, "\n");
}
strcat (clangVersion, "\n\n"); /* end of the comp! */

/*****
Now that we know what how many of each of the types of
SIGNALS or faultMsg we can declare their data structures.
*****/
strcat (compHeader, "\n\tchar faultString[120];");
if (numberOfPrints)
{
    sprintf (tmpl, "\n\tstatic struct faultMsgStruct faultStruct[%d];", numberOfPrints);
    strcat (compHeader, tmpl);
}
if (sigStringArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigStringStruct sigString[%d];", sigStringArraySize);
    strcat (compHeader, tmpl);
}
if (sigShortArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigShortStruct sigShort[%d];", sigShortArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffShortArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffShortStruct sigOffShort[%d];", sigOffShortArraySize);
    strcat (compHeader, tmpl);
}
if (sigIntArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigIntStruct sigInt[%d];", sigIntArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffIntArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffIntStruct sigOffInt[%d];", sigOffIntArraySize);
    strcat (compHeader, tmpl);
}
if (sigFloatArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigFloatStruct sigFloat[%d];", sigFloatArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffFloatArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffFloatStruct sigOffFloat[%d];", sigOffFloatArraySize);
    strcat (compHeader, tmpl);
}
if (sigDoubleArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigDoubleStruct sigDouble[%d];", sigDoubleArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffDoubleArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffDoubleStruct sigOffDouble[%d];", sigOffDoubleArraySize);
    strcat (compHeader, tmpl);
}
if (sigLongArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigLongStruct sigLong[%d];", sigLongArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffLongArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigOffLongStruct sigOffLong[%d];", sigOffLongArraySize);
    strcat (compHeader, tmpl);
}
if (sigUnsignedArraySize)
{
    sprintf (tmpl, "\n\tstatic struct sigUnsignedStruct sigUnsigned[%d];", sigUnsignedArraySize);
    strcat (compHeader, tmpl);
}
if (sigOffUnsignedArraySize)
```

```

    sprintf(tmp1, "\n\tstatic struct sigOffUnsignedStruct sigOffUnsigned[%d];", sigOffUnsignedArraySize);
    strcat (compHeader, tmp1);
}

/*****
Loop through the entire variable list for the comp
*****/
for (i=0; i<numCompVars; i++)
{
    /*****
    If we find a local put its type declaration into
    the c string with appropriate indentation.
    *****/
    if (strcmp (CompVars[i].class, "local") == 0)
    {
        /*****
        Char strings are handled a little differently
        *****/
        if (CompVars[i].type[0] == 'c')
        {
            sprintf(tmp1, "\n\tstatic char %s[120];", CompVars[i].name);
        }
        else sprintf (tmp1, "\n\tstatic %s %s;", CompVars[i].type, CompVars[i].name);
        strcat (compHeader, tmp1);
    }
}

/*****
Add a carriage return to lift and seperate (pronounce with a long 'a')
*****/
strcat (compHeader, "\n");

/*****
Let up now retrieve the signals necessary to run this comp
*****/
for (i=0; i<numCompVars; i++)
{
    /*****
    If we find a signal look at its type and cat the
    appropriate get_sig call into the header definition
    *****/
    if (strcmp (CompVars[i].class, "signal") == 0)
    {
        if (CompVars[i].put_or_get == GET || CompVars[i].put_or_get == PET)
        {
            if (CompVars[i].type[0] == 'c')
                sprintf (tmp1, "\n\tgetsig (\042%s\042, %s);", CompVars[i].name, CompVars[i].name);
            else sprintf (tmp1, "\n\tgetsig (\042%s\042, %s);", CompVars[i].name, CompVars[i].name);

            strcat (compHeader, tmp1);
        }
    }
}

/*****
Add a carriage return to lift and seperate (pronounce with a long 'a')
*****/
strcat (compHeader, "\n");

/*****
FILE I/O
Now that we have completed the clangVersion we need to
write it to its file so it can be used by install.
*****/
sprintf (compName_c, "%s/%s/%s.c", CodeGroups, GroupName, compName);
if (!(ptr = fopen (compName_c, "w", "translate")))
{
    sprintf(msgString, "Translate: CODE could not open %s for writing, aborting translation...", compName_c);
    user_ack(msgString, HELP_U_ACK);
    return (ERROR);
}

/*****
Write the clangVersion into its C file for install to use.
*****/
fprintf (ptr, "%s", compHeader, clangVersion);
fclose (ptr);

```

```
    } /* end of translate() */

    /*****
    yankBlank & yankToken functions
    *****/
    yankBlank (comp, compIndex)
    char *comp;
    int *compIndex;
    {
        /*****
        Yank non-chars until you hit a char
        *****/
        while ((comp[*compIndex] == ' ' || comp[*compIndex] == '\t' ||
            comp[*compIndex] == '\n') && comp[*compIndex] != 0)
        {
            (*compIndex)++;
        }
    }

    yankToken (comp, compIndex, token, tokenIndex)
    char *comp;
    int *compIndex;
    char *token;
    int *tokenIndex;
    {
        /*****
        Init the token to null
        *****/
        *tokenIndex = 0;
        /*****
        Yank char until you hit a non char
        *****/
        while (comp[*compIndex] != ' ' && comp[*compIndex] != '\t' &&
            comp[*compIndex] != '\n' && comp[*compIndex] != 0)
        {
            token[( *tokenIndex )++] = comp[( *compIndex )++];
        }
        token[*tokenIndex] = 0;
    }
}
```

90/06/07
13:14:35

type_check.c

1

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"

/*****<----->*****/
*
* MODULE NAME: type_check( type )
*
*
* MODULE FUNCTION:
*
* Checks if the specified data type is valid at the current location in
* the comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Terri Murphy & Troy Heindel
* NASA/JSC/MOD
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

type_check( type )
char type;
{
    int illegal = FALSE;

    switch(CompareType[NumberOfCompares-1][0])
    {
        case 's':    if(WhereAmI == CONSEQUENCE && type != 's' ||
                       WhereAmI == PREMISE && type == 'c')
                       illegal = TRUE;
                       break;

        case 'f':    if((WhereAmI == CONSEQUENCE &&
                           (type == 'c' || type == 'd' || type == 'h')) ||
                       (WhereAmI == PREMISE && type == 'c'))
                       illegal = TRUE;
                       break;

        case 'i':    if((WhereAmI == CONSEQUENCE &&
                           (type == 'c' || type == 'd' || type == 'f' || type == 'h')) ||
                       (WhereAmI == PREMISE && type == 'c'))
                       illegal = TRUE;
                       break;

        case 'c':    if(WhereAmI == PREMISE && type != 'c')
                       illegal = TRUE;
                       break;

        case 'd':    if(type == 'c')
                       illegal = TRUE;
                       break;

        case 'h':    /* Need to know how translate defines hex types */
                       break;

        default:     break;
    }

    if (illegal)
    {
        return(ERROR);
    }

    if (CompareType[NumberOfCompares-1][0] != type)
```

90/06/07
13:14:35

type_check.c

2

```
{  
    user_ack ("Warning: Operand types are inconsistent in this expression.", HELP_U_ACK);  
}  
return(OK);
```

90/06/07
13:14:38

utilities.c

1

```

/*****<----->*****/
*
* FILE NAME:    utilities.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which initialize the X Windows connections and
*   define the X Windows widgets.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   cleanExit()      - kills the X Window connection and exits the Comp Builder
*   cleanSlate()     - initializes the variables used by a Comp
*   get_type()       - determines the data type of currently selected Comp var
*   get_type_gc()    - determines if user wants to del/create a comp or group
*   getVarIndex()    - determines the data type of the variable
*   hardcopy()       - prints the group files
*   indent()         - maintains the indentation and spacing in the comp string
*   load_font()      - loads a logical font
*   openFile()       - opens the specified file and performs error processing
*   readCODEFile()   - reads in the CODE file into a given string
*   readCompNames()  - extracts the names of the comps for the current group
*   readGroupNames() - extracts the names of the available groups
*   set_size()       - sets the Work Area font
*   str_isalnum()    - determines if a string is alphanumeric
*   writeGroupNames() - writes the group names back to the group name file
*
*****/

#include <stdio.h>
#include <ctype.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/MessageB.h>
#include "code.h"
#include "widgets.h"
```

90/06/07
13:14:38

utilities.c

2

```

/*****<----->*****/
*
* MODULE NAME: cleanExit ( void )
*
*
* MODULE FUNCTION:
*
*   Routine kills the X Window connection and exits the Comp Builder.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

```

```
void cleanExit()
{

```

```

    /*
    * Destroy the root application shell widget and thereby, all subordinate widgets which
    * make up the window and any popup windows used for menus.
    */

```

```
    XtDestroyWidget ( top );

```

```
    exit( ERROR );
}

```


90/06/07
13:14:38

utilities.c

3

```

/*****<----->*****/
*
* MODULE NAME: cleanSlate()
*
*
* MODULE FUNCTION:
*
*   Function is used by retrieve() and get_header() to initialize the global variables
*   used by a Comp.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel/NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void cleanSlate()
{
    ChoiceCounter = 0;
    ParenCount = 0;
    Equation = LHS;
    WhereAmI = CONSEQUENCE;
    NumberOfIfs = NumberOfEndifs = 0;
    NumberOfCompares = 0;
    NumCompVars = 0;
    FuncParenCount = 0;
    Comp[0] = 0;
}

```

90/06/07
13:14:38

utilities.c

4

```

/*****<----->*****/
*
* MODULE NAME:  get_type( whose_type, the_message )
*
*
* MODULE FUNCTION:
*
*   Processes the popup that determines the data type for the currently selected
*   Comp variable.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

int get_type( whose_type, the_message )
{
    char    *the_message,
            *whose_type;

    Arg args[1];

    XtSetArg( args[0], XmNlabelString, XmStringLtoRCreate(the_message,XmSTRING_DEFAULT_CHARSET) );
    XtSetValues( lbl_type, args, 1 );

    /*
     * Popup the widget
     */

    process_popup( dlg_type, WAIT );

    if ( Get_Type_Stat == ABORT )
        return( ABORT );
    else
        strcpy( whose_type, tokens[ Get_Type_Stat ].name );
}

```

```

/*****<----->*****/
*
* MODULE NAME:  get_type_gc( prompt )
*
*
* MODULE FUNCTION:
*
* Processes the popup that determines if the user wants to create/delete a group
* or a comp.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
*                   Data Systems Department
*                   Automation and Data Systems Division
*                   Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*****/

int get_type_gc( prompt )
{
    char *prompt;

    Arg args[1];

    XtSetArg( args[0], XmNlabelString, XmStringLtoRCreate(prompt,XmSTRING_DEFAULT_CHARSET) );
    XtSetValues( lbl_gc, args, 1 );

    process_popup( dlg_gc, WAIT );

    return( Get_GC_Stat );
}

/*****
    getVarIndex

Returns:

    0 - The given name is of class local or signal.
    -1 - The given name was not found in either the
         local or group variable lists.
index# -
If name was found as an msid in the group list
then an index is returned.
*****/

getVarIndex( name, CompVars, numCompVars, varIndex, relIndex )
{
    char *name;
    struct var_struct CompVars[]; /* Structure containing the comp variable defs */
    int numCompVars;              /* The number of comp variable in the structure */
    int *varIndex;                /* The index of the variable; MSID, SIGNAL, LOCAL */
    int *relIndex;                /* The relative index of the MSID */

    int i, msid_cnt;

    *relIndex = 0;
    *varIndex = 0;

    /*****
    If the first char is a double quote we know
    right away that this is a literal string.
    *****/
    if (name[0] == '"')
    {

```

90/06/07
13:14:38

utilities.c

6

```
        return (STRING);
    }

    /*****
    See if the variable is in the variable passed
    to translate() from install.
    *****/
    for(i=0;i<numCompVars;i++)
    {
        /*****
        Try to match the name, then the class
        *****/
        if (strcmp (CompVars[i].name, name) == 0)
        {
            if (strcmp (CompVars[i].class, "signal") == 0)
            {
                *varIndex = i;
                return(SIGNAL);
            }
            else if (strcmp (CompVars[i].class, "local") == 0)
            {
                *varIndex = i;
                return(LOCAL);
            }
            else if (strcmp (CompVars[i].class, "msid") == 0)
            {
                *varIndex = i;
                break;
            }
        }
    }

    /*****
    If we did not find it as a SIGNAL or LOCAL then see
    if it is an MSID in the CompInfo structure
    *****/
    msid_cnt = 0;
    for (i=0;i<NumGroupVars;i++)
    {
        /*****
        We need to keep track of a relative index for
        the msids.
        *****/
        if (strcmp (GroupVars[i].class, "msid") == 0)
        {
            /*****
            Try to match the name, return if so
            *****/
            if (strcmp (GroupVars[i].name, name) == 0)
            {
                *relIndex = msid_cnt; /* the relative msid index */
                return (MSID);
            }
            msid_cnt++;
        }
    }

    /*****
    Now we know that it was not an MSID, signal, or
    local, but now see if it was a number, or a string
    *****/
    if (str_isalnum (name) != ERROR)
    {
        return (NUMBER);
    }
    return(ERROR);
}

/*****
HARDCOPY
```

Purpose: Hardcopy sends the following files to the printing device: comp_file, variable file, the "C" version.

Designer: Troy Heindel

Programmer: Troy Heindel

90/06/07
13:14:38

utilities.c

7

Date: 5/7/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by: Terri Murphy

Reasons for Revision: IGP vs Sony print routines.

*****/

hardcopy ()

```
{
    char hard_copy[400],
    CODELanguage[PATH_LEN], /* Path of high level language file */
    compVariables[PATH_LEN], /* Path of comp variable file */
    compCLanguage[PATH_LEN], /* Path of comp C language file */
    groupCLanguage[PATH_LEN]; /* Path of group C language file */

    /*****
    Prepare the paths
    *****/
    sprintf (CODELanguage, "%s/%s/%s.h", CodeGroups, GroupName, CompName);
    sprintf (compVariables, "%s/%s/%s.v", CodeGroups, GroupName, CompName);
    sprintf (compCLanguage, "%s/%s/%s.c", CodeGroups, GroupName, CompName);
    sprintf (groupCLanguage, "%s/%s/%s.c", CodeGroups, GroupName, GroupName);

    /*****
    Prepare the string
    *****/
    sprintf (hard_copy, "lpr %s %s %s %s >>/tmp/code.err 2>&1", CODELanguage, compVariables,
        compCLanguage, groupCLanguage);
    /*****
    Ask UNIX for a little help
    *****/
    user_ack("Printing...", HELP_U_ACK);
    system (hard_copy);
}
```

90/06/07
13:14:38

utilities.c

8

```

/*****<----->*****/
*
* MODULE NAME:  indent( theArea )
*
*
* MODULE FUNCTION:
*
*   Maintains the indentation within the Comp string.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel/NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

```

indent( theArea )
{
    int theArea;

    int    i;

    /*
     *   Put the correct number of spaces for the area we are in
     */

    if (theArea == PREMISE)
    {
        /*
         *   If we are following some logic we need a carriage
         *   return and spaces, otherwise just a single space
         */

        if ( PrevChoice[ChoiceCounter-1] == AND      ||
             PrevChoice[ChoiceCounter-1] == OR       ||
             PrevChoice[ChoiceCounter-1] == BITOR    ||
             PrevChoice[ChoiceCounter-1] == BITAND   ||
             PrevChoice[ChoiceCounter-1] == BITXOR )
        {
            strcat( Comp, "\n" );
            updateWA( "\n" );
            for (i=0 ; i < ((NumberOfIfs-NumberOfEndifs)-1)*SIZE_INDENT+3; i++ )
            {
                strcat( Comp, " " );
                updateWA( " " );
            }
        }
        else
        {
            strcat( Comp, " " );
            updateWA( " " );
        }
    }
    else /* CONSEQUENCE */
    {
        /*
         *   If we are following a THEN or an ELSE then
         *   we need a return and spaces, otherwise just a

```

90/06/07
13:14:38

utilities.c

9

```
* single space
*/
if ( PrevChoice[ChoiceCounter-1] != THEN &&
    PrevChoice[ChoiceCounter-1] != ELSE)
{
    strcat (Comp, "\n");
    updateWA( "\n" );
    for ( i=0; i < (NumberOfIfs-NumberOfEndifs)*SIZE_INDENT; i++ )
    {
        strcat( Comp, " " );
        updateWA( " " );
    }
}
else
{
    strcat( Comp, " " );
    updateWA( " " );
}
}
```

```

/*****<----->*****/
*
* MODULE NAME:  load_font( font, font structure )
*
*
* MODULE FUNCTION:
*
*   Routine loads an XmFontList structure based on the logical character name
*   of the specified font.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

int load_font( font, fnt_list )

    char      *font;
    XmFontList *fnt_list;

{
    XFontStruct *font_struct = NULL;

    font_struct = XLoadQueryFont( XtDisplay(top), font );
    if (! font_struct)
    {
        select_cursor( Shuttle_Cursor );
        user_ack( "Couldn't XLoad font", HELP_U_ACK );
        return( ERROR );
    }

    *fnt_list = XmFontListCreate( font_struct, XmSTRING_DEFAULT_CHARSET );

    return( OK );
}
```


90/06/07
13:14:38

utilities.c

11

```

/*****<----->*****/
*
* MODULE NAME:  openFile( file, option, callingRoutine )
*
*
* MODULE FUNCTION:
*
*   Function opens the specified file with the specified options and performs
*   error processing.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel/NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

```

FILE *openFile(file, option, callingRoutine)

```

    char    *callingRoutine,
            *file,
            *option;

    {
        FILE    *filePtr;

        char    explain[250],
                tmpString[22];

        if ( ! (filePtr = fopen( file, option)) )
        {
            switch ( option[0] )
            {
                case 'r' : strcpy( tmpString, "reading" );
                            break;
                case 'a' : strcpy( tmpString, "appending" );
                            break;
                case 'w' : strcpy( tmpString, "writing" );
                            break;
                default  : strcpy( tmpString, "bad open file option" );
            }
            sprintf( explain, "Can not open %s for %s", file, tmpString );
            select_cursor( Shuttle_Cursor );
            user_ack( explain, HELP_U_ACK );
            return( NULL );
        }
        else
            return( filePtr );
    }

```

```

/*****
    readCODEFile

```

Purpose: readCODEFile reads in the CODE high level file into the string which is passed to it.

Designer: Troy Heindel/NASA/JSC/MOD

Programmer: Troy Heindel/NASA/JSC/MOD

Date: 1/24/89

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/

readCODEFile( compName, compStr )
    char compName[]; /* The name of the comp to get */
    char *compStr; /* The string to put the comp into */
{
    FILE *filePtr; /* You guessed it! */

    int i;

    char theCODEFile[PATH_LEN]; /* The path to the CODE file */

    /*****
    Create the path to open the high level file
    *****/
    sprintf (theCODEFile, "%s/%s/%s.h", CodeGroups, GroupName, compName);

    /*****
    Open the GroupInfo file with the option
    *****/
    if (!(filePtr = openFile (theCODEFile, "r", "readCODEFile")))
        return (ERROR);

    /*****
    Read the entire comp into the comp string.
    *****/
    i=0;
    while (((compStr[i++] = fgetc (filePtr)) != EOF) && i<MAX_COMP_LEN);
    fclose (filePtr);
    compStr[i-1] = '\0';

    /*****
    Make sure we're not too big.
    *****/
    if(i==MAX_COMP_LEN)
    {
        user_ack("This comp exceeds the maximum comp length. Please notify developers.", HELP_U_ACK);
        return (ERROR);
    }

    return (OK);
}
```

90/06/07
13:14:38

utilities.c

13

```

/*****<----->*****/
*
* MODULE NAME:  readCompNames()
*
*
* MODULE FUNCTION:
*
*   Extracts the names of the comp files for the current group.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel/NASA/JSC/MOD
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

```

```
int readCompNames()
{
    FILE *filePtr; /* You guessed it! */

    char file_str[PATH_LEN];

    int i,j;

    NumOfComps = 0;
    /*
     * Let's open the [GroupName].dat file to find out
     * which comps exist for this group.
     */
    sprintf (file_str, "%s/%s.dat", AMSupport, GroupName);
    if (!(filePtr = openFile (file_str, "r", "readCompNames")))
        return (NumOfComps);
    /*
     * ... and read the CompName, noise_filter, rate,
     * on_off and disposition into an array of group
     * information structures.
     */
    else
    {
        /*
         * Throw out the header comment.
         */
        fscanf(filePtr, "%*[^\n]");
        /*
         * Read in what's left.
         */
        while ((fscanf (filePtr, "%s %d %d %d %d %d %d %[^\n]",
            CompInfo[NumOfComps].name,
            &CompInfo[NumOfComps].noise_filter,
            &CompInfo[NumOfComps].rate,
            &CompInfo[NumOfComps].on_off,
            &CompInfo[NumOfComps].disposition,
            &CompInfo[NumOfComps].cycle_mode,
            CompInfo[NumOfComps].purpose)) != EOF)
        {
            NumOfComps++;
        }
        fclose (filePtr);
        /*****
        Sort the little buggers so they're nice to look at.
        *****/
    }
}
```

90/06/07
13:14:38

utilities.c

14

```
*****/
for (i=0;i<NumOfComps-1;i++)
{
    for (j=i+1;j<NumOfComps;j++)
    {
        if (strcmp (CompInfo[i].name, CompInfo[j].name) > 0)
        {
            CompInfo[MAX_COMPS] = CompInfo[i];
            CompInfo[i]          = CompInfo[j];
            CompInfo[j]          = CompInfo[MAX_COMPS];
        }
    }
}
return (NumOfComps);
}
```

```
/*----->*****  
*  
* MODULE NAME: readGroupNames()  
*  
*  
* MODULE FUNCTION:  
*  
* Extracts the names of the available groups.  
*  
*  
* SPECIFICATION DOCUMENTS:  
*  
* /code/specs/code  
*  
*  
* ORIGINAL AUTHOR AND IDENTIFICATION:  
*  
* Troy Heindel/NASA/JSC/MOD  
*  
* Timothy J. Barton - Software Engineering Section  
* Data Systems Department  
* Automation and Data Systems Division  
* Southwest Research Institute  
*  
*  
* REVISION HISTORY:  
*  
* Motif Release 1.0 - 90/03/16  
*  
*****<----->*/
```

```
int readGroupNames()  
{  
    FILE *filePtr; /* You guessed it! */  
    char message[120];  
    struct group_info_struct tempGroupInfo;  
    int i,j; /* counters */  
    NumOfGroups = 0;  
    /*  
    * Open the GroupNames file for reading  
    */  
    if (filePtr = openFile (GroupNamesFile, "r", "readGroupNames"))  
    {  
        /*  
        * Strip the header.  
        */  
        fscanf(filePtr, "%*[^\\n]");  
        while (fscanf (filePtr, "%s %d", GroupInfo[NumOfGroups].name,  
                        &GroupInfo[NumOfGroups].disposition) != EOF)  
            ++NumOfGroups;  
        fclose (filePtr);  
        /*  
        * Sort the little buggers so they're nice to look at.  
        */  
        for (i=0;i<NumOfGroups-1;i++)  
        {  
            for (j=i+1;j<NumOfGroups;j++)  
            {  
                if (strcmp (GroupInfo[i].name, GroupInfo[j].name) > 0)  
                {  
                    tempGroupInfo = GroupInfo[i];  
                    GroupInfo[i] = GroupInfo[j];  
                    GroupInfo[j] = tempGroupInfo;  
                }  
            }  
        }  
    }  
    return (NumOfGroups);  
}
```

```
/*  
 * Maybe this is a new system and the file does not exist yet.  
 * Try creating the file by opening it for writing.  
 */  
  
else if (!(filePtr = openFile (GroupNamesFile, "w", "readGroupNames")))  
{  
    sprintf( message, "Unable to access the %s file in order to obtain", GroupNamesFile );  
    strcat ( message, " a listing of the groups.  " );  
    user_ack( message, HELP_U_ACK );  
    return (ERROR);  
}  
else  
{  
    user_ack("No algorithms were found on this machine", HELP_U_ACK);  
    fclose (filePtr);  
    return (0); /* The number of Groups on this system */  
}  
}
```

```

/*****<----->*****/
*
* MODULE NAME:  set_size()
*
*
* MODULE FUNCTION:
*
*   Sets the Work Area font based on the user's selection.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

```
void set_size()
{
    Arg          args[2];
    XmFontList   font;

    select_cursor( Clock_Cursor );

    set_font_tgl();

    /*
     * Set Work Area attributes.
     */

    switch( WorkA )
    {
        case DEFAULT :
            if ( load_font(Font_Default, &font) == ERROR )
                return;
            break;

        case OPTION1_FONT :
            if ( load_font(Font_Option1, &font) == ERROR )
                return;
            break;

        case OPTION2_FONT :
            if ( load_font(Font_Option2, &font) == ERROR )
                return;
            break;
    }

    XtSetArg( args[0], XmNfontList, (XtArgVal) font );
    XtSetValues( txt_worka, args, 1 );

    select_cursor( Shuttle_Cursor );
}
```

```

/*****
    STR_ISALNUM

Purpose:  Determines whether a given string is alphanumeric.
         A single decimal point is allowed.

Returns:  0 - It was alphanumeric.
         1 - It was not!
*****/
```

Designer: Troy Heindel/NASA/MOD

Programmer: Troy Heindel/NASA/MOD

Date: 11/14/88

Version: 2.0

Project: CODE (Comp Development Environment)

```
*****/
str_isalnum ( theString )
char *theString; /* The string to look at */
{
    int i,          /* Can't live without an i */
        decimalFlag = FALSE; /* TRUE if we've seen a decimal point */

    /*****
    Leading zero's are a no-no as they conote octal
    *****/
    if (theString[0] == '0' && strlen(theString) > 1)
    {
        user_ack("Leading zeros are not allowed", HELP_U_ACK);
        return (ERROR);
    }
    /*****
    Loop through the whole bloody string, checking to
    see that each char is an alphanumeric
    *****/
    for (i=0; i<strlen(theString); i++)
    {
        /*****
        Check to see if the char is an alphanumeric
        *****/
        if (!((theString[i] >= 'A' && theString[i] <= 'F') ||
            (theString[i] >= '0' && theString[i] <= '9'))))
        {
            /*****
            No. Then check to see if it is a decmial point.
            Only let them use one decmial point per number
            *****/
            if (theString[i] == '.' && decimalFlag == FALSE)
                decimalFlag = TRUE;
            else return (ERROR);
        }
    }
    if (decimalFlag)
        return (FLOAT); /* All's well that ends well */
    return (INTEGER); /* All's well that ends well */
}
```



```

/*****<----->*****/
*
* MODULE NAME:  writeGroupNames()
*
*
* MODULE FUNCTION:
*
*   Writes the group names back to the group name file.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Troy Heindel/NASA/JSC/MOD
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

int writeGroupNames()
{
    FILE *filePtr; /* You guessed it! */

    int i;

    /*****
     * Open the GroupInfo file with the option
     *****/
    if (!(filePtr = openFile (GroupNamesFile, "w", "writeGroupNames")))
        return (ERROR);

    /*****
     * On successful open, write the file
     *****/
    fprintf (filePtr, "%sGroupNames Disposition\n");

    for (i=0; i<NumOfGroups; i++)
        fprintf (filePtr, "%s %d\n", GroupInfo[i].name, GroupInfo[i].disposition);
    fclose (filePtr);

    return (OK);
}
```

90/06/07
13:14:41

var_check.c

1

VAR_CHECK

Purpose: Var_check checks the variable list to determine whether the variable just input has been previously defined.

Designer: Troy A. Heindel/NASA/JSC/MOD

Programmer: Troy A. Heindel/NASA/JSC/MOD

Date: 12/11/86

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

None.

Global Variables

NumCompVars The number of variables in this comp.

Include files

*****/
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "code.h"

90/06/07
13:14:41

var_check.c

2

```
var_check (name, local_index, global_index, class)
char name[]; /* The name of the passed variable */
int *local_index; /* The index into the CompVars struct */
int *global_index; /* The index into the MSIDTable struct */
char class[]; /* Either msid, signal, or local */

{
    int i; /* Counter */
    int defined_globally;
    int defined_locally;

    /*****
     Assume the worst--undefined variable
     *****/
    defined_globally = FALSE;
    defined_locally = FALSE;
    *local_index = -1;
    *global_index = -1;

    /*****
     Look for the msid in the struct MSIDTable
     *****/
    if (strcmp (class, "msid") == 0)
    {
        /*****
         If there is not MSIDTable then we can't look it up
         *****/
        if (!(MSIDTable[0].name))
            user_ack("\n\tMSID's will not be verified; the msid table was not found", HELP_U_ACK);
        else
        {
            /*****
             Try to find the msid in the tag_MSIDTable
             *****/
            for (i=0; i < MSIDCount; i++)
            {
                if (strcmp (name, MSIDTable[i].name) == 0)
                {
                    defined_globally = TRUE;
                    *global_index = i;
                    break;
                }
            }
            /*****
             If we did not find the msid in the msid list
             it is undefined and not usable.
             *****/
            if (!(defined_globally))
            {
                *global_index = ERROR;
            }
        }
    }

    /*****
     Look for signals in the struct SignalTable
     *****/
    else if (strcmp (class, "signal") == 0)
    {
        /*****
         Look for the signal in the SignalTable list
         *****/
        for (i=0; i < SignalCount; i++)
        {
            if (strcmp (name, SignalTable[i].name) == 0)
            {
                defined_globally = TRUE;
                *global_index = i;
                break;
            }
        }
    }

    /*****
     Always check to see whether the variable
     is defined locally to this comp.
     *****/
    for (i=0; i < NumCompVars; i++)
    {
        if (strcmp (name, CompVars[i].name) == 0)
    }
}
```

90/06/07
13:14:41

var_check.c

3

```
(
/******
Return an ERROR if they are trying to use an msid
as a local, signal or something of the sort
*****/
if ((strcmp (class, CompVars[i].class) == 0) || (strcmp(class,"null") == 0))
{
    *local_index = i;
    defined_locally = TRUE;
    break;
}
else /* It is a miss-classing e.g. a local as a signal */
{
    return (ERROR);
}
}
return (OK); /* The variable is ok */
)
```

var_check.c

```

****
funcCheck
****
ncCheck checks to see if a given unknown token is
d in the function list. Return 1 if true, 0
wise.

Troy A. Heindel/NASA/JSC/MOD
Troy A. Heindel/NASA/JSC/MOD
5/88
2.0
CODE (Comp Development Environment)

or Revision:
-----
Interfaces

****
k (funcName)          /* The name of the professed function */
funcName[];

i;

** *****
y ***** nd the passed funcName in the list of
er ***** ined functions, return OK if found
*****
: (i=0;i<NumberOfUserFuncs;i++)
if (strcmp (funcName, UserFuncs[i]) == 0)
return (OK);

*****
return an ERROR if it wasn't a function.
*****
return (ERROR);

```

ORIGINAL PAGE IS
OF POOR QUALITY

90/06/07
13:14:43

widgets.h

2

scr_gname,

sep_ack, sep_actn, sep_arch, sep_ask, sep_dev1, sep_dev2, sep_file,
sep_fnt, sep_fnt2, sep_gc, sep_gname, sep_gstr, sep_hello, sep_help,
sep_help_txt, sep_logic, sep_math, sep_oper, sep_type, sep_var, separator1,

tgl_lshot, tgl_cycle, tgl_dev1, tgl_dev2, tgl_dev3, tgl_wa_de, tgl_wa_o1, tgl_wa_o2,

txt_gstr_reply, txt_file, txt_status, txt_worka;

90/06/07
13:14:46

window_io.c

1

```
/*-----*/
*
* FILE NAME:      window_io.c
*
*
* FILE FUNCTION:
*
*   Contains the functions which prompt the user and display data to the user.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   ask                () - processes the popup which asks a yes/no question
*   clear_inform       () - takes down the inform() popup
*   display_file       () - opens a disk file and displays it to the user in a popup
*   display_str        () - displays a large string to the user via a popup
*   displayWA          () - display a string to the user in the Work Area
*   get_string         () - prompts the user to enter a string, processes the popup
*   inform             () - displays a message to the user
*   set_device_tgl     () - sets the state of the backup device toggles
*   set_font_tgl       () - sets the state of the Work Area font selection toggles
*   updateWA           () - adds a token to the Comp currently in the Work Area
*   user_ack           () - displays a message to the user in a popup
*
*-----*/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/MwmUtil.h>
#include <Xm/MessageB.h>
#include <Xm/Text.h>
#include "code.h"
#include "widgets.h"

extern XtCallbackRec clear_popup[];

Arg args[4];
```

90/06/07
13:14:46

window_io.c

2

```

/*****<----->*****/
*
* MODULE NAME:  ask( prompt )
*
*
* MODULE FUNCTION:
*
*   Processes the popup which asks the user a specified yes/no question.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

int ask( prompt )
{
    char    *prompt;

    {
        /*
         * Setup the prompt string.
         */

        XtSetArg( args[0], XmNlabelString, XmStringLtoRCreate(prompt,XmSTRING_DEFAULT_CHARSET) );
        XtSetValues( lbl_ask, args, 1 );

        /*
         * Set the size (width) of the prompt.
         */

        if ( SetNum == 1 )
            XtSetArg( args[0], XmNwidth, (150+(5*strlen(prompt))) );
        else
            XtSetArg( args[0], XmNwidth, (200+(10*strlen(prompt))) );

        XtSetValues( frm_ask, args, 1 );

        process_popup( dlg_ask, WAIT );
        return( Ask_Resp );
    }
}
```


90/06/07
13:14:46

window_io.c

3

```

/*****<---->*****/
*
* MODULE NAME:  clear_inform()
*
*
* MODULE FUNCTION:
*
*   Function when working (which it's not), will clear the inform() popup.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<---->*****/

void clear_inform()
{
/*DEBUG
   XtUnmanageChild( dlg_inform );
*/
}
```

90/06/07
13:14:46

window_io.c

4

```
*****<---->*****
*
* MODULE NAME:  display_file( display_type, fileToDisplay )
*
*
* MODULE FUNCTION:
*
*   Reads a file and places a scrolled window popup to display the file to the
*   user.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****<---->*****/
```

```
void display_file( display_type, fileToDisplay )

    int    display_type;
    char    *fileToDisplay;

{
    FILE        *fp;

    register int    i    = 0,
                    ptr = 0;

    char          c,
                    string[101];

    select_cursor( Clock_Cursor );

    /*
     * Open the file.
     */

    if ( (fp = openFile( fileToDisplay, "r", "display_file" )) == NULL )
        return;

    /*
     * Read data from the file. Read 100 bytes at a time and add to the text widget's
     * string.
     */

    XmTextSetString( txt_file, NULLS );
    while ( ptr != EOF )
    {
        while ( i < 100 && ( string[ i ] = c = getc ( fp ) ) != EOF )
            i++;
        string[ i ] = NULL;
        XmTextReplace ( txt_file, ptr, ptr, string );
        if ( c == EOF )
            ptr = EOF;
        else
        {
            ptr += i;
            i = 0;
        }
    }

    /*
     * Close the file and setup the help string.
     */
}
```

C. 6

```
*/  
  
fclose( fp );  
  
switch ( display_type )  
{  
    case HELP    : XtSetArg( args[1], XmNlabelString,  
                             XmStringLtoRCreate( Help_Txt_Str, XmSTRING_DEFAULT_CHARSET ));  
                   break;  
    case LIST    : XtSetArg( args[1], XmNlabelString,  
                             XmStringLtoRCreate( List_Txt_Str, XmSTRING_DEFAULT_CHARSET ));  
                   break;  
}  
  
/*  
 * Set the font of the label and text widgets.  
*/  
  
XtSetArg( args[0], XmNfontList, Fnt_List_Btn );  
XtSetValues( lbl_help_txt, args, 2 );  
XtSetValues( txt_file,    args, 1 );  
  
select_cursor( Shuttle_Cursor );  
process_popup( dlg_file, WAIT );  
}
```

```

/*****<----->*****/
*
* MODULE NAME:  display_str( display_type, stringToDisplay )
*
*
* MODULE FUNCTION:
*
*   Displays a large string to the use via a scrolled window popup.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void display_str( display_type, strToDisplay )

    int      display_type;
    char      *strToDisplay;

{
    XmTextPosition ptr = XmTextGetLastPosition( txt_file );

    /*
     * Insert the new string.
     */

    XmTextReplace( txt_file, 0, ptr, strToDisplay );

    switch ( display_type )
    {
        case HELP : XtSetArg( args[1], XmNlabelString,
                               XmStringLtoRCreate( Help_Txt_Str, XmSTRING_DEFAULT_CHARSET ));
                    break;
        case LIST  : XtSetArg( args[1], XmNlabelString,
                               XmStringLtoRCreate( List_Txt_Str, XmSTRING_DEFAULT_CHARSET ));
                    break;
    }

    /*
     * Set the font of the label and text widgets.
     */

    XtSetArg( args[0], XmNfontList, Fnt_List_Btn );
    XtSetValues( lbl_help_txt, args, 2 );
    XtSetValues( txt_file, args, 1 );

    select_cursor( Shuttle_Cursor );
    process_popup( dlg_file, WAIT );
}
```

90/06/07
13:14:46

window_io.c

7

```

/*****<----->*****/
*
* MODULE NAME:  displayWA( stringToDisplay )
*
*
* MODULE FUNCTION:
*
*   Routine inserts the specified text string into the Work Area text widget for
*   display to the user.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

```

void displayWA( strToDisplay )
{
    char *strToDisplay;

    {
        select_cursor( Clock_Cursor );
        XmTextSetString( txt_worka, strToDisplay );
        select_cursor( Shuttle_Cursor );
    }
}
```

90/06/07
13:14:46

window_io.c

8

```

/*****<----->*****/
*
* MODULE NAME:  get_string( prompt, answer, answer_size, multi_line )
*
*
* MODULE FUNCTION:
*
*   Processes the popup which prompts the user to enter a string.  The string can
*   be either a single line string (filename, variable name) or it can be multi-
*   line (Comp Purpose).
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

int get_string( prompt, answer, answer_size, multi_line )

    char    *answer,
            *prompt;
    int     answer_size,
            multi_line;

    char          *tmp;
    int           rc;

/*
 * If the response from the user can be multi-line, then setup the
 * text widget and button defaults.
 */

if ( multi_line )
{
    XtSetArg( args[0], XmNshowAsDefault, FALSE );
    XtSetValues( btn_gok, args, 1 );

    XtSetArg( args[0], XmNactivateCallback, NULL );
    XtSetArg( args[1], XmNeditMode, XmMULTI_LINE_EDIT );
    XtSetArg( args[2], XmNrows, 5 );
    XtSetArg( args[3], XmNcolumns, 80 );
}

/*
 * The user's response can only be a single line, setup the text
 * widget and button defaults.
 */

else
{
    XtSetArg( args[0], XmNshowAsDefault, TRUE );
    XtSetValues( btn_gok, args, 1 );

    XtSetArg( args[0], XmNdefaultButton, btn_gok );
    XtSetValues( dlg_gstr, args, 1 );

    clear_popup[0].closure = (caddr_t) G_OK;
    XtSetArg( args[0], XmNactivateCallback, clear_popup );
    XtSetArg( args[1], XmNeditMode, XmSINGLE_LINE_EDIT );
    XtSetArg( args[2], XmNrows, 1 );
    XtSetArg( args[3], XmNcolumns, 80 );
}
```

```
    XtSetValues      ( txt_gstr_reply, args, 4 );
    XmTextSetMaxLength( txt_gstr_reply, answer_size );
    XmTextSetString   ( txt_gstr_reply, NULLS );

    /*
     * Set the size of the get_string popup.
     */

    XtSetArg( args[0], XmNheight, 250 );
    XtSetArg( args[1], XmNwidth, 500 );
    XtSetValues( frm_gstr, args, 2 );

    XtSetArg( args[0], XmNlabelString, XmStringLtoRCreate(prompt,XmSTRING_DEFAULT_CHARSET) );
    XtSetValues( lbl_gstr, args, 1 );

    process_popup( dlg_gstr, WAIT );

    /*
     * Check to ensure the user didn't chose to abort, get the
     * string entered by the user from the widget.
     */

    if ( Get_Str_Stat == ACCEPT )
    {
        tmp = XmTextGetString( txt_gstr_reply );
        strcpy( answer, tmp );
        XtFree( tmp );
        rc = ACCEPT;
    }
    else
        rc = ABORT;

    XtUnmanageChild( dlg_gstr );
    return( rc );
}
```

```

/*****<----->*****/
*
* MODULE NAME:  inform( message )
*
*
* MODULE FUNCTION:
*
*   Displays a message to the user.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void inform( message )

    char *message;

{
/*
    int         count = 0;
    XmString    tcs;
    XEvent      event;
    tcs = XmStringLtoRCreate( message, XmSTRING_DEFAULT_CHARSET );

    XtSetArg( args[0], XmNmessageString, tcs );
    XtSetArg( args[1], XmNdialogStyle, XmDIALOG_APPLICATION_MODAL );
    XtSetValues ( dlg_inform, args, 2 );
    XtManageChild( dlg_inform );
    XmStringFree ( tcs );

    while ( count++ < 5 )
    {
        XtNextEvent( &event );
        XtDispatchEvent( &event );
    }
*/
}

```



```
/*-----*/
*
* MODULE NAME:  set_device_tgl()
*
*
* MODULE FUNCTION:
*
*   Sets the state of the backup device toggle buttons.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/

void set_device_tgl()
{
    disarm_toggle( tgl_dev1 );
    disarm_toggle( tgl_dev2 );
    disarm_toggle( tgl_dev3 );

    switch ( Device )
    {
        case DEV1: arm_toggle( tgl_dev1 );
                   break;
        case DEV2: arm_toggle( tgl_dev2 );
                   break;
        case DEV3: arm_toggle( tgl_dev3 );
                   break;
    }
}
```

90/06/07
13:14:46

window_io.c

12

```
/*-----*/
*
* MODULE NAME:  set_font_tgl()
*
*
* MODULE FUNCTION:
*
*   Sets the state of the Work Area font selection toggle buttons.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
void set_font_tgl()
{
    disarm_toggle( tgl_wa_de );
    disarm_toggle( tgl_wa_o1 );
    disarm_toggle( tgl_wa_o2 );

    switch ( WorkA )
    {
        case DEFAULT      : arm_toggle( tgl_wa_de );
                           break;
        case OPTION1_FONT : arm_toggle( tgl_wa_o1 );
                           break;
        case OPTION2_FONT : arm_toggle( tgl_wa_o2 );
                           break;
    }
}
```

```
/*-----*/
*
* MODULE NAME:  updateWA( strToAppend )
*
*
* MODULE FUNCTION:
*
*   Adds a new token to the Comp currently displayed in the Work Area.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
void updateWA( strToAppend )
{
    char *strToAppend;

    XmTextPosition ptr;

    ptr = XmTextGetLastPosition( txt_worka );
    XmTextReplace( txt_worka, ptr, ptr, strToAppend );
    ptr = XmTextGetLastPosition( txt_worka );
    XmTextShowPosition( txt_worka, ptr );
}
```

```
/*----->*****
*
* MODULE NAME:  user_ack( message, help_num )
*
*
* MODULE FUNCTION:
*
*   Displays a message to the user and waits for an acknowledgement from the user
*   that the message was read.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****<----->*/

void user_ack( message, help_num )

    char    *message;
    int      help_num;

{
    /*
     * Set the context-sensitive help selection.
     */

    PopupHelp = help_num;

    /*
     * Setup up the message to display to the user.
     */

    XtSetArg( args[0], XmNlabelString, XmStringLtoRCreate(message,XmSTRING_DEFAULT_CHARSET) );
    XtSetValues( lbl_ack, args, 1 );

    /*
     * Set the size (width) of the user_ack() popup.  This is a pretty ugly
     * method, but it works.
     */

    if ( SetNum == 1 )
        XtSetArg( args[0], XmNwidth, (150+(5*strlen(message))) );
    else
        XtSetArg( args[0], XmNwidth, (200+(8*strlen(message))) );
    XtSetValues( frm_ack, args, 1 );

    select_cursor( Shuttle_Cursor );
    process_popup( dlg_ack, WAIT );
}
```

90/06/07
14:19:20

init_gp.c

1

```
*****<----->*****
*
* FILE NAME:    init_gp.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which initialize the X Windows connections and
*   define the X Windows widgets.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   build_get_name()    - builds the get_name() popup
*   get_defaults()      - gets the defaults for sizes, fonts, colors, etc.
*   init_graphics()     - builds all the widgets and popups, opens connection to X
*   name_to_pixel()     - converts a textual name to an X pixel value
*   select_cursor()     - selects the specified cursor shape
*   set_btn_defaults()  - sets the defaults
*   set_defaults()      - sets defaults for the popups and main screen forms
*
*****<----->*****/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/cursorfont.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/mwm.h>
#include <Xm/Form.h>
#include <Xm/MainW.h>
#include <Xm/DialogS.h>
#include <Xm/MessageB.h>
#include <Xm/RowColumn.h>
#include "code.h"
#include "create.h"
#include "widgets.h"
#include "pixmaps.h"
#include "cbr_popups.h"

/*
 * File globals.
 */

Arg      args[10];

char      Comp_Str   [200],
          Group_Str  [200],

          *hello_str,
          *name_str,
          dev1_str[40],
          dev2_str[40],
          dev3_str[40],
          de_str  [20],
          ol_str  [20],
          o2_str  [20],

          font_dsp_str[] = "abcdef ABCDEF 1234567890",

          help_str[] = "Select the button to display HELP for",
          tmp_str1[] = "  Name                Disposition",
          tmp_str2[] = "    ";

/*
 * Function prototypes.
 */

Pixel  name_to_pixel();
```

```

/*****<----->*****/
*
* MODULE NAME:  build_get_name()
*
* MODULE FUNCTION:
*
*   This routine builds the get comp/group name popup.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void build_get_name( )
{
    /*
     * Build the two strings, one for groups and one for comps.
     */

    strcpy( Group_Str, "Listing of Available Groups\n\n\n" );
    strcat( Group_Str, tmp_str1 );  strcat( Group_Str, tmp_str2 );
    strcat( Group_Str, tmp_str1 );  strcat( Group_Str, tmp_str2 );
    strcat( Group_Str, tmp_str1 );

    strcpy( Comp_Str, "Listing of Available Comps\n\n\n" );
    strcat( Comp_Str, tmp_str1 );  strcat( Comp_Str, tmp_str2 );
    strcat( Comp_Str, tmp_str1 );  strcat( Comp_Str, tmp_str2 );
    strcat( Comp_Str, tmp_str1 );

    /*
     * Build the popup dialog and related form and label widgets.
     */

    dlg_gname = cr_popup( NULLS, top );
    frm_gname = cr_form( "frm_gname", dlg_gname, NULL, NULL );
    lbl_gname = cr_label( "lbl_gname", frm_gname, Group_Str, 0, 0, 18, 0, 100 );

    /*
     * Build the scrolled window widget and row column widget which will contain
     * a button for each of the comp/group names.
     */

    XtSetArg( args[0], XmNtopAttachment,      XmATTACH_POSITION );
    XtSetArg( args[1], XmNtopPosition,        20 );
    XtSetArg( args[2], XmNscrollBarDisplayPolicy, XmAS_NEEDED );
    XtSetArg( args[3], XmNscrollingPolicy,    XmAUTOMATIC );
    XtSetArg( args[4], XmNrightAttachment,    XmATTACH_FORM );
    XtSetArg( args[5], XmNleftAttachment,    XmATTACH_FORM );
    XtSetArg( args[6], XmNbottomAttachment,   XmATTACH_POSITION );
    XtSetArg( args[7], XmNbottomPosition,    82 );
    XtSetArg( args[8], XmNvisualPolicy,      XmCONSTANT );
    scr_gname = XmCreateScrolledWindow( frm_gname, "scr_gname", args, 9 );
    XtManageChild( scr_gname );

    rcl_gname = cr_rowcol( NULLS, scr_gname, 3, XmVERTICAL );
    XtSetArg ( args[0], XmNworkWindow, rcl_gname );
    XtSetValues( scr_gname, args, 1 );

    /*

```

```
 * Build the separator and control buttons.  
 */
```

```
sep_gname = cr_separator(NULLS,frm_gname, 87, 89, 0, 100 );  
btn_gname_c = cr_rel_cmd (NULLS,frm_gname,"Cancel",clear_get_name, ABORT, 92, 15);  
btn_gname_h = cr_rel_cmd (NULLS,frm_gname,"Help", clear_get_name, GN_HELP, 93, 70);  
  
set_btn_defaults( btn_gname_c, DEFAULT );  
set_btn_defaults( btn_gname_h, FALSE );  
  
XtSetArg( args[0], XmNdefaultButton, btn_gname_c );  
XtSetValues( dlg_gname, args, 1 );  
  
XtSetArg( args[0], XmNlabelString, XmStringLtoRCreate( Group_Str, XmSTRING_DEFAULT_CHARSET ));  
XtSetValues( lbl_gname, args, 1 );  
  
XtSetArg( args[0], XmNtitle, " " );  
XtSetValues( XtParent( dlg_gname ), args, 1 );  
}
```

```

/*****<---->*****/
*
* MODULE NAME:  get_defaults()
*
* MODULE FUNCTION:
*
*   Determines the defaults for the fonts, sizes, and colors of the X Windows
*   widgets.
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*****/
```

```

void get_defaults()
{
    char    *set_name;

    /*
     * Get the insensitive token color, if not found, set the default anyway.
     */
    ELIColor = XGetDefault(display,name_str,"ELIColor");
    if ( ELIColor == NULL )
    {
        ELIColor = (char *) malloc(sizeof(char) * 20 );
        strcpy( ELIColor, "lightgray" );
    }

    /*
     * Get the default fonts, if not found, set them anyway.
     */
    strcpy( Font_Default, "fixed" );

    Font_Option1 = XGetDefault(display,name_str,"OptFont1");
    if ( Font_Option1 == NULL )
    {
        Font_Option1 = (char *) malloc(sizeof(char) * 10 );
        strcpy( Font_Option1, "6x10" );
    }

    Font_Option2 = XGetDefault(display,name_str,"OptFont2");
    if ( Font_Option2 == NULL )
    {
        Font_Option2 = (char *) malloc(sizeof(char) * 10 );
        strcpy( Font_Option2, "9x15" );
    }

    /*
     * Get/set the button, status area, and work area sizes and fonts for
     * Set1.
     */
    set_name = XGetDefault( display, name_str, "SetName" );
    if ((set_name == NULL) || (!strcmp(set_name,"Set1",4)))
    {
        SetNum = 1;

        BT_Width = 50;
        EL_Width = 250;
    }
}
```


90/06/07
14:19:20

init_gp.c

5

```
FM_Width = atoi(XGetDefault(display,name_str,"FM_Width_Set1"));

BT_Height = 20;
EL_Height = atoi(XGetDefault(display,name_str,"EL_Height_Set1"));
ST_Height = atoi(XGetDefault(display,name_str,"ST_Height_Set1"));
WA_Height = atoi(XGetDefault(display,name_str,"WA_Height_Set1"));

strcpy( BT_Font, "fixed" );
strcpy( EL_Font, "fixed" );

ST_Font = XGetDefault(display,name_str,"ST_Font_Set1");
WA_Font = XGetDefault(display,name_str,"WA_Font_Set1");
}

/*
 * Get/set the button, status area, and work area sizes and fonts for
 * Set2.
 */

else
{
    SetNum = 2;

    BT_Width = 75;
    EL_Width = 350;
    FM_Width = atoi(XGetDefault(display,name_str,"FM_Width_Set2"));

    BT_Height = 30;
    EL_Height = atoi(XGetDefault(display,name_str,"EL_Height_Set2"));
    ST_Height = atoi(XGetDefault(display,name_str,"ST_Height_Set2"));
    WA_Height = atoi(XGetDefault(display,name_str,"WA_Height_Set2"));

    strcpy( BT_Font, "9x15" );
    strcpy( EL_Font, "9x15" );

    ST_Font = XGetDefault(display,name_str,"ST_Font_Set2");
    WA_Font = XGetDefault(display,name_str,"WA_Font_Set2");
}

/*
 * Setup font of push buttons.
 */

load_font( BT_Font, &Fnt_List_Btn );
}
```

```
/*----->*****  
*  
* MODULE NAME:  init_graphics()  
*  
*  
* MODULE FUNCTION:  
*  
* This function initializes the connection to the X server and builds all of the main  
* screen widgets.  This functions calls functions to set the defaults for the widgets  
* and then makes a call to X to Realize the main screen.  
*  
*  
* SPECIFICATION DOCUMENTS:  
*  
* /code/specs/code  
*  
* ORIGINAL AUTHOR AND IDENTIFICATION:  
*  
* Timothy J. Barton - Software Engineering Section  
* Data Systems Department  
* Automation and Data Systems Division  
* Southwest Research Institute  
*  
* REVISION HISTORY:  
*  
* Motif Release 1.0 - 90/03/16  
*  
*****<----->*****/  
  
void init_graphics( argc, argv )  
  
    int      argc;  
    char     **argv;  
  
    {  
        Pixel    background,  
                foreground;  
  
        Pixmap    pixmap;  
  
        XImage *image;  
  
        Widget    frm_elmnt;  
  
        XtToolkitInitialize();  
        display = XtOpenDisplay( NULL, NULL, argv[0], "CODE", NULL, 0, &argc, argv );  
        if (! display )  
        {  
            XtWarning("CODE: Can't open display.");  
            exit(0);  
        }  
  
        name_str = argv[0];  
  
        /*  
        * Create top level shell.  
        */  
  
        top = XtAppCreateShell( argv[0], "CODE", applicationShellWidgetClass, display, NULL, 0 );  
        XtManageChild( top );  
  
        win_main = XmCreateMainWindow( top, "win_main", NULL, 0 );  
        XtManageChild( win_main );  
  
        /*  
        * Determine defaults.  
        */  
  
        get_defaults();  
  
        /*  
        * Create the main window menu bar.  
        */
```

```
mnb_main = XmCreateMenuBar( win_main, "mnb_main", NULL, 0 );
XtManageChild( mnb_main );

/*
 * Create the main window pull down menu.
 */

mnu_file = XmCreatePulldownMenu( mnb_main, "mnu_file", NULL, 0 );
mnu_edit = XmCreatePulldownMenu( mnb_main, "mnu_edit", NULL, 0 );
mnu_list = XmCreatePulldownMenu( mnb_main, "mnu_list", NULL, 0 );
mnu_opt  = XmCreatePulldownMenu( mnb_main, "mnu_opt",  NULL, 0 );
mnu_help = XmCreatePulldownMenu( mnb_main, "mnu_help", NULL, 0 );

btn_file = cr_cascade( NULLS, mnb_main, mnu_file, 'F', "File" );
btn_edit = cr_cascade( NULLS, mnb_main, mnu_edit, 'E', "Edit" );
btn_listf = cr_cascade( NULLS, mnb_main, mnu_list, 'L', "List" );
btn_opt   = cr_cascade( NULLS, mnb_main, mnu_opt,  'O', "Options" );
btn_help  = cr_cascade( NULLS, mnb_main, mnu_help, 'H', "Help" );

w[CREATE] = cr_command( NULLS, mnu_file, "Create", code_menu, CREATE );
w[SAVE]   = cr_command( NULLS, mnu_file, "Save",   code_menu, SAVE );
w[COPY]   = cr_command( NULLS, mnu_file, "Copy",   code_menu, COPY );
w[PRINT]  = cr_command( NULLS, mnu_file, "Print",  code_menu, HARDCOPY );
w[BACKUP] = cr_command( NULLS, mnu_file, "Backup", code_menu, BACKUP );
w[REMOVE] = cr_command( NULLS, mnu_file, "Remove", code_menu, REMOVE );
w[INSTALL] = cr_command( NULLS, mnu_file, "Install", code_menu, INSTALL );
w[RETRIEVE] = cr_command( NULLS, mnu_file, "Retrieve", code_menu, RETRIEVE );
separator1 = cr_separator( NULLS, mnu_file, IGNORE, IGNORE, IGNORE, IGNORE );
w[QUIT]    = cr_command( NULLS, mnu_file, "Quit",   code_menu, QUIT );

/*
 * Initialize mnemonics for selected action buttons.
 */

XtSetArg( args[0], XmNmnemonic, 'S' );
XtSetArg( args[1], XmNacceleratorText, XmStringCreateLtoR("F6", XmSTRING_DEFAULT_CHARSET));
XtSetArg( args[2], XmNaccelerator, "<Key>F6:" );
XtSetValues( w[SAVE], args, 3 );

XtSetArg( args[0], XmNmnemonic, 'I' );
XtSetArg( args[1], XmNacceleratorText, XmStringCreateLtoR("F5", XmSTRING_DEFAULT_CHARSET));
XtSetArg( args[2], XmNaccelerator, "<Key>F5:" );
XtSetValues( w[INSTALL], args, 3 );

XtSetArg( args[0], XmNmnemonic, 'R' );
XtSetArg( args[1], XmNacceleratorText, XmStringCreateLtoR("F2", XmSTRING_DEFAULT_CHARSET));
XtSetArg( args[2], XmNaccelerator, "<Key>F2:" );
XtSetValues( w[RETRIEVE], args, 3 );

XtSetArg( args[0], XmNmnemonic, 'Q' );
XtSetArg( args[1], XmNacceleratorText, XmStringCreateLtoR("F1", XmSTRING_DEFAULT_CHARSET));
XtSetArg( args[2], XmNaccelerator, "<Key>F1:" );
XtSetValues( w[QUIT], args, 3 );

/*
 * Build remaining action buttons.
 */

btn_cycle = cr_command( NULLS, mnu_list, "Comp Cycle Modes", code_menu, LIST_CYCLE );
btn_MSID  = cr_command( NULLS, mnu_list, "MSIDs",           code_menu, LIST_MSID );
btn_signl = cr_command( NULLS, mnu_list, "Signals",         code_menu, LIST_SIGNAL );
btn_funct = cr_command( NULLS, mnu_list, "User Functions",   code_menu, LIST_FUNCT );

btn_font  = cr_command( NULLS, mnu_opt, "Set Fonts",         code_menu, FONTS );
btn_device = cr_command( NULLS, mnu_opt, "Set Backup Device", code_menu, DEVICE );

w[DELETE] = cr_command( NULLS, mnu_edit, "Delete",          code_menu, DELETE );
btn_vi    = cr_command( NULLS, mnu_edit, "vi",              code_menu, EDIT );
btn_tokens = cr_command( NULLS, mnu_help, "Token Help",      code_menu, HELP_TOKENS );
btn_manual = cr_command( NULLS, mnu_help, "Browse Manual",    code_menu, HELP_MANUAL );

/*
 * Initialize mnemonics for remaining action buttons.
 */

XtSetArg( args[0], XmNmnemonic, 'D' );
XtSetArg( args[1], XmNacceleratorText, XmStringCreateLtoR("F3", XmSTRING_DEFAULT_CHARSET));
XtSetArg( args[2], XmNaccelerator, "<Key>F3:" );
```

```
XtSetValues( w[DELETE], args, 3 );

XtSetArg( args[0], XmNmemonic, 'v');
XtSetArg( args[1], XmNacceleratorText, XmStringCreateLtoR("F4", XmSTRING_DEFAULT_CHARSET));
XtSetArg( args[2], XmNaccelerator, "<Key>F4:");
XtSetValues( btn_vi, args, 3 );

/*
 * Build Comp Tokens.
 */

form      = cr_form( "form", win_main, NULL, NULL );
frame     = cr_frame( "frame", form, NULL, NULL );

frm_elmnt = cr_form( NULLS, frame, NULL, NULL );
frm_funcnt = cr_form( "frm_funcnt", frm_elmnt, NULL, NULL );
frm_opt   = cr_form( "frm_opt", frm_elmnt, NULL, NULL );
lbl_funcnt = cr_label( "lbl_funcnt", frm_funcnt, "COMP Elements", 1, 1, 7, 1, 97 );
lbl_logic  = cr_label( "lbl_logic", frm_funcnt, "Logic", 0, 9, 12, 1, 99 );
lbl_var    = cr_label( "lbl_var", frm_funcnt, "Variable", 0, 28, 31, 1, 99 );
lbl_oper   = cr_label( "lbl_oper", frm_funcnt, "Relational", 0, 41, 44, 1, 99 );
lbl_actn   = cr_label( "lbl_actn", frm_funcnt, "Action", 0, 54, 57, 1, 99 );
lbl_math   = cr_label( "lbl_math", frm_funcnt, "Math", 0, 72, 75, 1, 99 );

lbl_opt    = cr_label( "lbl_opt", frm_opt, "COMP Options", 1, 1, 40, 1, 97 );
tgl_cycle  = cr_toggle( "tgl_cycle", frm_opt, "Cyclic", code_menu, TGL_CYCLE, TGL_CYCLE );
tgl_1shot  = cr_toggle( "tgl_1shot", frm_opt, "One-Shot", code_menu, TGL_1SHOT, TGL_1SHOT );

sep_logic  = cr_separator( NULLS, frm_funcnt, 13, 14, 1, 97 );
rcl_logic  = cr_rowcol( "rcl_logic", frm_funcnt, 2, XmHORIZONTAL );
w[IF]      = cr_command( NULLS, rcl_logic, "if", code_menu, IF );
w[THEN]    = cr_command( NULLS, rcl_logic, "then", code_menu, THEN );
w[ELSE]    = cr_command( NULLS, rcl_logic, "else", code_menu, ELSE );
w[END_IF]  = cr_command( NULLS, rcl_logic, "endif", code_menu, END_IF );
w[AND]     = cr_command( NULLS, rcl_logic, "and", code_menu, AND );
w[OR]      = cr_command( NULLS, rcl_logic, "or", code_menu, OR );
w[NOT]     = cr_command( NULLS, rcl_logic, "not", code_menu, NOT );
w[BITAND]  = cr_command( NULLS, rcl_logic, "bitAnd", code_menu, BITAND );
w[BITOR]   = cr_command( NULLS, rcl_logic, "bitOr", code_menu, BITOR );
w[BITXOR]  = cr_command( NULLS, rcl_logic, "bitXor", code_menu, BITXOR );

sep_var    = cr_separator( NULLS, frm_funcnt, 32, 33, 1, 97 );
rcl_var    = cr_rowcol( "rcl_var", frm_funcnt, 1, XmHORIZONTAL );
w[MSID]    = cr_command( NULLS, rcl_var, "MSID", code_menu, MSID );
w[SIGNAL]  = cr_command( NULLS, rcl_var, "signal", code_menu, SIGNAL );
w[LOCAL]   = cr_command( NULLS, rcl_var, "local", code_menu, LOCAL );
w[NUMBER]  = cr_command( NULLS, rcl_var, "number", code_menu, NUMBER );
w[STRING]  = cr_command( NULLS, rcl_var, "string", code_menu, STRING );

sep_oper   = cr_separator( NULLS, frm_funcnt, 45, 46, 1, 97 );
rcl_oper   = cr_rowcol( "rcl_oper", frm_funcnt, 1, XmHORIZONTAL );
w[EQ]      = cr_command( NULLS, rcl_oper, "=", code_menu, EQ );
w[NE]      = cr_command( NULLS, rcl_oper, "<", code_menu, NE );
w[LE]      = cr_command( NULLS, rcl_oper, "<=", code_menu, LE );
w[GE]      = cr_command( NULLS, rcl_oper, ">=", code_menu, GE );
w[LT]      = cr_command( NULLS, rcl_oper, "< ", code_menu, LT );
w[GT]      = cr_command( NULLS, rcl_oper, "> ", code_menu, GT );

sep_actn   = cr_separator( NULLS, frm_funcnt, 58, 59, 1, 97 );
rcl_actn   = cr_rowcol( "rcl_actn", frm_funcnt, 2, XmHORIZONTAL );
w[SET]     = cr_command( NULLS, rcl_actn, "set", code_menu, SET );
w[PRINT]   = cr_command( NULLS, rcl_actn, "print", code_menu, PRINT );
w[FUNCTION] = cr_command( NULLS, rcl_actn, "function", code_menu, FUNCTION );
w[COMMENT] = cr_command( NULLS, rcl_actn, "comment", code_menu, COMMENT );
w[START]   = cr_command( NULLS, rcl_actn, "start", code_menu, START );
w[STOP]    = cr_command( NULLS, rcl_actn, "stop", code_menu, STOP );

sep_math   = cr_separator( NULLS, frm_funcnt, 76, 77, 1, 97 );
rcl_math   = cr_rowcol( "rcl_math", frm_funcnt, 4, XmHORIZONTAL );
w[ADD]     = cr_command( NULLS, rcl_math, "+", code_menu, ADD );
w[SUBTRACT] = cr_command( NULLS, rcl_math, "-", code_menu, SUBTRACT );
w[MULTIPLY] = cr_command( NULLS, rcl_math, "*", code_menu, MULTIPLY );
w[DIVIDE]  = cr_command( NULLS, rcl_math, "/", code_menu, DIVIDE );
w[L_PAREN] = cr_command( NULLS, rcl_math, "(", code_menu, L_PAREN );
w[R_PAREN] = cr_command( NULLS, rcl_math, ")", code_menu, R_PAREN );
w[COMMA]   = cr_command( NULLS, rcl_math, ",", code_menu, COMMA );
w[PI]      = cr_command( NULLS, rcl_math, "PI", code_menu, PI );
```

```
w[SQRT] = cr_command( NULLS, rcl_math, "sqrt", code_menu, SQRT);
w[POWER] = cr_command( NULLS, rcl_math, "power", code_menu, POWER);
w[EXP] = cr_command( NULLS, rcl_math, "exp", code_menu, EXP);
w[SHIFTL] = cr_command( NULLS, rcl_math, "shiftL", code_menu, SHIFTL);
w[SHIFTR] = cr_command( NULLS, rcl_math, "shiftR", code_menu, SHIFTR);
w[LOG] = cr_command( NULLS, rcl_math, "log", code_menu, LOG);
w[COS] = cr_command( NULLS, rcl_math, "cos", code_menu, COS);
w[SIN] = cr_command( NULLS, rcl_math, "sin", code_menu, SIN);
w[TAN] = cr_command( NULLS, rcl_math, "tan", code_menu, TAN);
w[ACOS] = cr_command( NULLS, rcl_math, "acos", code_menu, ACOS);
w[ASIN] = cr_command( NULLS, rcl_math, "asin", code_menu, ASIN);
w[ATAN] = cr_command( NULLS, rcl_math, "atan", code_menu, ATAN);

txt_status = cr_text("txt_status", form, frame, NULL, NULLS, NO_SCROLL, 1, 80 );
txt_worka = cr_text("txt_worka", form, frame, txt_status, NULLS, SCROLL_BARS, 25, 80 );

/*
 * Build ask() popup dialog.
 */

dlg_ask = cr_popup( NULLS, top );
frm_ask = cr_form ( "frm_ask", dlg_ask, NULL, NULL );

image = CreateDefaultImage (questionBits, 22, 22);
XmInstallImage (image, "question_img");
XtSetArg( args[0], XmNforeground, &foreground );
XtSetArg( args[1], XmNbackground, &background );
XtGetValues( dlg_ask, args, 2 );

pixmap = XmGetPixmap( XtScreen(dlg_ask), "question_img", foreground, background );

pix_ask = cr_pixmap ( "pix_ask", frm_ask, &pixmap, 0, IGNORE, 1, IGNORE );
lbl_ask = cr_label ( "lbl_ask", frm_ask, NULLS, 0, 0, IGNORE, 0, 100 );
sep_ask = cr_separator( NULLS, frm_ask, 40, 45, 0, 100 );

btn_ask_y = cr_rel_cmd("btn_ask_y", frm_ask, "YES", clear_popup, ASK_YES, 61, 15);
btn_ask_n = cr_rel_cmd("btn_ask_n", frm_ask, "NO", clear_popup, ASK_NO, 65, 45);
btn_ask_h = cr_rel_cmd("btn_ask_h", frm_ask, "Help", show_help, ASK_HELP, 65, 70);

/*
 * Build user_ack() popup dialog.
 */

dlg_ack = cr_popup( NULLS, top );
frm_ack = cr_form ( "frm_ack", dlg_ack, NULL, NULL );

image = CreateDefaultImage (infoBits, 11, 24);
XmInstallImage (image, "info_img");

XtSetArg( args[0], XmNforeground, &foreground );
XtSetArg( args[1], XmNbackground, &background );
XtGetValues( dlg_ack, args, 2 );

pixmap = XmGetPixmap( XtScreen(dlg_ack), "info_img", foreground, background );

pix_ack = cr_pixmap ( "pix_ack", frm_ack, &pixmap, 0, IGNORE, 1, IGNORE );
lbl_ack = cr_label ( "lbl_ack", frm_ack, NULLS, 0, 0, IGNORE, 0, 100 );
sep_ack = cr_separator( NULLS, frm_ack, 40, 45, 0, 100 );

btn_ack_y = cr_rel_cmd("btn_ack_y", frm_ack, "OK", clear_popup, USER_ACK, 61, 15);
btn_ack_h = cr_rel_cmd("btn_ack_h", frm_ack, "Help", show_help, USER_HELP, 63, 70);

/*
 * Build group/comp selection popup dialog.
 */

dlg_gc = cr_popup( NULLS, top );
frm_gc = cr_form ( "frm_gc", dlg_gc, NULL, NULL );

XtSetArg( args[0], XmNforeground, &foreground );
XtSetArg( args[1], XmNbackground, &background );
XtGetValues( dlg_gc, args, 2 );

pixmap = XmGetPixmap( XtScreen(dlg_gc), "question_img", foreground, background );

pix_gc = cr_pixmap ( "pix_gc", frm_gc, &pixmap, 5, IGNORE, 1, IGNORE );
lbl_gc = cr_label ( "lbl_gc", frm_gc, NULLS, 0, 0, 40, 5, 100 );
sep_gc = cr_separator( NULLS, frm_gc, 45, 50, 0, 100 );
```

```
btn_group = cr_rel_cmd("btn_group", frm_gc, "Group", clear_popup, GC_GROUP, 71, 10 );
btn_comp = cr_rel_cmd("btn_comp", frm_gc, "Comp", clear_popup, GC_COMP, 73, 30 );
btn_gcc = cr_rel_cmd("btn_gcc", frm_gc, "Cancel", clear_popup, GC_CANCEL, 73, 50 );
btn_gch = cr_rel_cmd("btn_gch", frm_gc, "Help", show_help, GC_HELP, 73, 70 );

/*
 * Build archive function popup dialog.
 */

dlg_arch = cr_popup( NULLS, top );
frm_arch = cr_form ( "frm_arch", dlg_arch, NULL, NULL );

XtSetArg( args[0], XmNforeground, &foreground );
XtSetArg( args[1], XmNbackground, &background );
XtGetValues( dlg_arch, args, 2 );

pixmap = XmGetPixmap( XtScreen(dlg_arch), "question_img", foreground, background );

pix_arch = cr_pixmap ( "pix_arch", frm_arch, &pixmap, 5, IGNORE, 1, IGNORE );
lbl_arch = cr_label ( "lbl_arch", frm_arch, NULLS, 0, 0, IGNORE, 5, 100 );
sep_arch = cr_separator( NULLS, frm_arch, 55, 60, 0, 100 );
btn_read = cr_rel_cmd( NULLS, frm_arch, "Read", clear_popup, ARCH_READ, 71, 12 );
btn_list = cr_rel_cmd( NULLS, frm_arch, "List", clear_popup, ARCH_LIST, 74, 30 );
btn_write = cr_rel_cmd( NULLS, frm_arch, "Write", clear_popup, ARCH_WRITE, 74, 45 );
btn_cancel = cr_rel_cmd( NULLS, frm_arch, "Cancel", clear_popup, ARCH_CANCEL, 74, 60 );
btn_ahelp = cr_rel_cmd( NULLS, frm_arch, "Help", show_help, ARCH_HELP, 74, 75 );

/*
 * Build the get_name() popup dialog.
 */

build_get_name();

/*
 * Build the display_file() popup dialog.
 */

dlg_file = cr_popup( NULLS, top );
frm_file = cr_form ( "frm_file", dlg_file, NULL, NULL );
txt_file = cr_text ( NULLS, frm_file, NULL, NULL, "", SCROLL_BARS, 20, 80 );
sep_file = cr_separator( NULLS, frm_file, 87, 89, 0, 100 );
btn_ok = cr_rel_cmd ( NULLS, frm_file, "Close", clear_popup, FILE_OK, 92, 15 );
btn_fhelp = cr_rel_cmd ( NULLS, frm_file, "Help", show_help, FILE_HELP, 93, 70 );

/*
 * Build the get_string() popup dialog.
 */

dlg_gstr = cr_popup( NULLS, top );
frm_gstr = cr_form ( "frm_gstr", dlg_gstr, NULL, NULL );
lbl_gstr = cr_label( "lbl_gstr", frm_gstr, NULLS, 0, 0, IGNORE, 0, 100 );
txt_gstr_reply = cr_frm_txt( "txt_gstr_reply", frm_gstr, "", NULL, NULL, NO_SCROLL, 5, 80 );

sep_gstr = cr_separator( NULLS, frm_gstr, 70, 75, 0, 100 );

btn_gok = cr_rel_cmd( "btn_gok", frm_gstr, "Ok", clear_popup, G_OK, 81, 15 );
btn_gcancl = cr_rel_cmd( "btn_gcancl", frm_gstr, "Cancel", clear_popup, G_CANCL, 83, 45 );
btn_ghelp = cr_rel_cmd( "btn_ghelp", frm_gstr, "Help", show_help, G_HELP, 83, 70 );

/*
 * Build the get_type() popup dialog.
 */

dlg_type = cr_popup( NULLS, top );
frm_type = cr_form ( "frm_type", dlg_type, NULL, NULL );

XtSetArg( args[0], XmNforeground, &foreground );
XtSetArg( args[1], XmNbackground, &background );
XtGetValues( dlg_type, args, 2 );

pixmap = XmGetPixmap( XtScreen(dlg_type), "question_img", foreground, background );

pix_type = cr_pixmap ( "pix_type", frm_type, &pixmap, 5, IGNORE, 1, IGNORE );
lbl_type = cr_label ( "lbl_type", frm_type, NULLS, 0, 0, IGNORE, 5, 100 );
sep_type = cr_separator( NULLS, frm_type, 55, 60, 0, 100 );

btn_tcancl = cr_rel_cmd( NULLS, frm_type, "Cancel", clear_popup, T_CANCL, 71, 3 );
```

init_gp.c

```
btn_int    = cr_rel_cmd( NULLS, frm_type, "Int",    clear_popup, INTEGER, 74, 20);
btn_float  = cr_rel_cmd( NULLS, frm_type, "Float",  clear_popup, FLOAT,   74, 34);
btn_double = cr_rel_cmd( NULLS, frm_type, "Double", clear_popup, DOUBLE,  74, 48);
btn_char   = cr_rel_cmd( NULLS, frm_type, "Char",   clear_popup, CHAR,    74, 62);
btn_short  = cr_rel_cmd( NULLS, frm_type, "Short",  clear_popup, SHORT,   74, 76);
btn_thelp  = cr_rel_cmd( NULLS, frm_type, "Help",   show_help,  T_HELP,  74, 90);

/*
 * Build help() popup dialog.
 */

dlg_help   = cr_popup( NULLS, top );
frm_help   = cr_form ( "frm_help", dlg_help, NULL, NULL );

XtSetArg( args[0], XmNforeground, &foreground );
XtSetArg( args[1], XmNbackground, &background );
XtGetValues( dlg_help, args, 2 );

pixmap     = XmGetPixmap( XtScreen(dlg_help), "question_img", foreground, background );

pix_help   = cr_pixmap ( "pix_help", frm_help, &pixmap, 5, IGNORE, 1, IGNORE );
lbl_help   = cr_label ( "lbl_help", frm_help, help_str, 0, 10, IGNORE, 5, 100 );
sep_help   = cr_separator( NULLS, frm_help, help, 55, 60, 0, 100 );

btn_hlp_c  = cr_rel_cmd( "btn_help_y", frm_help, "Cancel", clear_popup, HELP_CANCEL, 71, 15);
btn_hlp_h  = cr_rel_cmd( "btn_help_n", frm_help, "Help",   show_help,  HELP_HELP,  75, 70);

/*
 * Build help text popup dialog.
 */

dlg_help_txt = cr_popup ( NULLS, top );
frm_help_txt = cr_form ( "frm_help_txt", dlg_help_txt, NULL, NULL );
lbl_help_txt = cr_label ( "lbl_help_txt", frm_help_txt, NULLS, 0, 0, IGNORE, 0, 100 );
sep_help_txt = cr_separator( NULLS, frm_help_txt, 70, 75, 0, 100 );
btn_help_txt = cr_rel_cmd ( NULLS, frm_help_txt, "Close", clear_popup, HELP_CLOSE, 81, 39);

/*
 * Build the font selection popup.
 */

strcpy( de_str, "Default:  " );
strcpy( ol_str, "Option 1:  " );
strcpy( o2_str, "Option 2:  " );

strcat( de_str, Font_Default );
strcat( ol_str, Font_Option1 );
strcat( o2_str, Font_Option2 );

dlg_font    = cr_popup ( NULLS, top );
frm_font    = cr_form ( "frm_font", dlg_font, NULL, NULL );
lbl_fonts   = cr_label ( "lbl_fonts", frm_font, "Set Work Area Font", 0, 0, 20, 0, 100 );
sep_fnt     = cr_separator( NULLS, frm_font, 22, 25, 5, 95 );

tgl_wa_de   = cr_toggle( "tgl_wa_de", frm_font, de_str, clear_popup, TGL_WA_DE, TGL_WA_DE );
tgl_wa_ol   = cr_toggle( "tgl_wa_ol", frm_font, ol_str, clear_popup, TGL_WA_OL, TGL_WA_OL );
tgl_wa_o2   = cr_toggle( "tgl_wa_o2", frm_font, o2_str, clear_popup, TGL_WA_O2, TGL_WA_O2 );

lbl_fnt_de  = cr_label ( "lbl_fnt_de", frm_font, font_dsp_str, 0, 29, 38, 31, 95 );
lbl_fnt_ol  = cr_label ( "lbl_fnt_ol", frm_font, font_dsp_str, 0, 43, 52, 31, 95 );
lbl_fnt_o2  = cr_label ( "lbl_fnt_o2", frm_font, font_dsp_str, 0, 57, 67, 31, 95 );
sep_fnt2    = cr_separator( NULLS, frm_font, 71, 74, 0, 100 );

btn_fn_ok   = cr_rel_cmd( "btn_fn_ok", frm_font, "Apply", clear_popup, FONT_OK, 83, 15);
btn_fn_cnl  = cr_rel_cmd( "btn_fn_cnl", frm_font, "Cancel", clear_popup, FONT_CANCEL, 84, 45);
btn_fn_hlp  = cr_rel_cmd( "btn_fn_hlp", frm_font, "Help", show_help, FONT_HELP, 84, 71);

/*
 * Build the backup device selection popup.
 */

strcpy( dev1_str, "/dev/rflp - MASSCOMP floppy " );
strcpy( dev2_str, "/dev/rctp - MASSCOMP 1/4 inch tape" );
strcpy( dev3_str, "/dev/rst8 - SUN 1/4 inch tape" );

dlg_device  = cr_popup( NULLS, top );
frm_device  = cr_form ( "frm_device", dlg_device, NULL, NULL );
```

init_gp.c

```
lbl_device = cr_label("lbl_device", frm_device, "Set Backup Device", 0, 0, 20, 0, 100 );
sep_dev1   = cr_separator( NULLS,   frm_device, 22, 25, 5, 95 );

tgl_dev1   = cr_toggle("tgl_dev1", frm_device, dev1_str, clear_popup, TGL_DEV1, TGL_DEV1);
tgl_dev2   = cr_toggle("tgl_dev2", frm_device, dev2_str, clear_popup, TGL_DEV2, TGL_DEV2);
tgl_dev3   = cr_toggle("tgl_dev3", frm_device, dev3_str, clear_popup, TGL_DEV3, TGL_DEV3);
sep_dev2   = cr_separator( NULLS,   frm_device, 71, 74, 0, 100 );

btn_dev_ok  = cr_rel_cmd(NULLS,      frm_device, "Apply",  clear_popup, DEVICE_OK,      83, 15);
btn_dev_cnl = cr_rel_cmd(NULLS,      frm_device, "Cancel",  clear_popup, DEVICE_CANCEL, 84, 45);
btn_dev_hlp = cr_rel_cmd(NULLS,      frm_device, "Help",    show_help,  DEVICE_HELP,  84, 71);

/*
 * Build "hello" popup.
 */

dlg_hello = cr_popup( NULLS, top );
frm_hello = cr_form ("frm_hello", dlg_hello, NULL, NULL );

image = CreateDefaultImage(shuttle_bits, 64, 64);
XmInstallImage(image, "shuttle_img");

image = CreateDefaultImage(nasa_bits, 64, 64);
XmInstallImage(image, "nasa_img");

XtSetArg( args[0], XmNforeground, &foreground );
XtSetArg( args[1], XmNbackground, &background );
XtGetValues( dlg_hello, args, 2 );

pixmap = XmGetPixmap( XtScreen(dlg_hello), "shuttle_img", foreground, background );
pix_shuttle = cr_pixmap("pix_shuttle", frm_hello, &pixmap, 10, IGNORE, 4, IGNORE );

pixmap = XmGetPixmap( XtScreen(dlg_hello), "nasa_img", foreground, background );
pix_nasa = cr_pixmap("pix_nasa", frm_hello, &pixmap, 10, IGNORE, IGNORE, 96 );

lbl_hello_t = cr_label ("lbl_hello_t", frm_hello, NULLS, 0, 5, IGNORE, 15, 85 );
lbl_hello   = cr_label ("lbl_hello",   frm_hello, NULLS, 0, 20, IGNORE, 10, 90 );
sep_hello   = cr_separator(NULLS,      frm_hello, 77, 82, 0, 100 );
btn_hello   = cr_rel_cmd (NULLS,      frm_hello, "Begin", clear_popup, HELLO, 88, 45);

/*
 * Set defaults.
 */

set_defaults();

/*
 * Setup the valid choices.
 */

color_valid( PrevChoice[ChoiceCounter-1], 99 );

/*
 * Assign members of main window. Realize screen.
 */

XmMainWindowSetAreas( win_main, mnb_main, NULL, NULL, NULL, form );

XtRealizeWidget( dlg_gc      );
XtRealizeWidget( dlg_ack    );
XtRealizeWidget( dlg_ask    );
XtRealizeWidget( dlg_arch   );
XtRealizeWidget( dlg_file   );
XtRealizeWidget( dlg_font   );
XtRealizeWidget( dlg_gstr   );
XtRealizeWidget( dlg_help   );
XtRealizeWidget( dlg_type   );
XtRealizeWidget( dlg_gname   );
XtRealizeWidget( dlg_device );

XtRealizeWidget( top );

/*
 * Setup sizes and fonts.
 */

select_cursor( Shuttle_Cursor );
```



```
/*  
 * Setup toggles.  
 */
```

```
WorkA = DEFAULT;  
set_font_tgl();
```

```
Device = DEV1;  
set_device_tgl();
```

```
/*  
 * Display welcome popup.  
 */
```

```
XFlush( display );  
process_popup( dlg_hello, WAIT );  
}
```

90/06/07
14:19:20

init_gp.c

14

```

/*****<----->*****/
*
* MODULE NAME:  name_to_pixel( color string )
*
*
* MODULE FUNCTION:
*
*   Converts a textual name of a color to a Pixel value.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/

Pixel name_to_pixel( name )

    char *name;

{
    Colormap    colormap;
    XColor      color;

    colormap = XDefaultColormap( display, DefaultScreen(display) );

    XParseColor( display, colormap, name, &color );
    XAllocColor( display, colormap, &color );

    return( color.pixel );
}

```

```

/*****<----->*****/
*
* MODULE NAME:  select_cursor( cursor )
*
*
* MODULE FUNCTION:
*
*   Changes the cursor to the specified shape.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void select_cursor( cursor )

    Cursor cursor;

{
    XDefineCursor( display, XtWindow(top),      cursor );
    XDefineCursor( display, XtWindow(dlg_gc),    cursor );
    XDefineCursor( display, XtWindow(dlg_ack),    cursor );
    XDefineCursor( display, XtWindow(dlg_ask),    cursor );
    XDefineCursor( display, XtWindow(dlg_arch),   cursor );
    XDefineCursor( display, XtWindow(dlg_file),   cursor );
    XDefineCursor( display, XtWindow(dlg_font),   cursor );
    XDefineCursor( display, XtWindow(dlg_gstr),   cursor );
    XDefineCursor( display, XtWindow(dlg_help),   cursor );
    XDefineCursor( display, XtWindow(dlg_type),   cursor );
    XDefineCursor( display, XtWindow(dlg_gname),  cursor );
    XDefineCursor( display, XtWindow(dlg_device), cursor );

    XFlush( display );
}
```

```

/*****<----->*****/
*
* MODULE NAME:  set_btn_defaults()
*
*
* MODULE FUNCTION:
*
*   Sets the defaults for the popup buttons.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void set_btn_defaults( widget, def_widget )

    Widget  widget;
    int     def_widget;

{
    XtSetArg( args[0], XmNwidth,    BT_Width );
    XtSetArg( args[1], XmNheight,   BT_Height );
    XtSetArg( args[2], XmNfontList, Fnt_List_Btn );

    /*
     * If the widget is the default for the popup, highlight it with a wider
     * boarder.
     */

    if ( def_widget )
    {
        XtSetArg( args[3], XmNshowAsDefault, TRUE );
        XtSetValues( widget, args, 4 );
    }
    else
        XtSetValues( widget, args, 3 );
}

```

```

/*****<----->*****/
*
* MODULE NAME:  set_defaults()
*
*
* MODULE FUNCTION:
*
*   This function sets the defaults for the main screen widgets and popup widgets.
*   Defaults such as size and position are set in this routine before the the
*   widgets are realized.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void set_defaults()
(
    /*
     * Setup cursors.
     */

    Shuttle_Cursor = XCreateFontCursor( display, XC_shuttle );
    Clock_Cursor   = XCreateFontCursor( display, XC_watch );
    Help_Cursor    = XCreateFontCursor( display, XC_question_arrow );

    XmSetMenuCursor( display, Shuttle_Cursor );

    /*
     * Setup fonts of font selection popup.
     */

    load_font( Font_Default, &Fnt_List );
    XtSetArg( args[0], XmNfontList, Fnt_List );
    XtSetValues( lbl_fnt_de, args, 1 );

    load_font( Font_Option1, &Fnt_List );
    XtSetArg( args[0], XmNfontList, Fnt_List );
    XtSetValues( lbl_fnt_ol, args, 1 );

    load_font( Font_Option2, &Fnt_List );
    XtSetArg( args[0], XmNfontList, Fnt_List );
    XtSetValues( lbl_fnt_o2, args, 1 );

    /*
     * Setup defaults of the Status Area.
     */

    load_font( ST_Font, &Fnt_List );

    XtSetArg( args[0], XmNeditable,          False );
    XtSetArg( args[1], XmNsensitive,         False );
    XtSetArg( args[2], XmNrightAttachment,   XmATTACH_POSITION );
    XtSetArg( args[3], XmNrightPosition,     99 );
    XtSetArg( args[4], XmNtopAttachment,     XmATTACH_POSITION );
    XtSetArg( args[5], XmNtopPosition,       1 );
    XtSetArg( args[6], XmNheight,            ST_Height );
    XtSetArg( args[7], XmNleftOffset,        5 );
    XtSetArg( args[8], XmNfontList,          Fnt_List );
    XtSetValues( txt_status, args, 9 );

    /*

```

```
/*
 * Setup defaults of the token selection frame.
 */

XtSetArg( args[0], XmNtopAttachment,      XmATTACH_POSITION );
XtSetArg( args[1], XmNtopPosition,        1 );
XtSetArg( args[2], XmNleftAttachment,      XmATTACH_POSITION );
XtSetArg( args[3], XmNleftPosition,        1 );
XtSetArg( args[4], XmNbottomOffset,        5 );
XtSetArg( args[5], XmNwidth,              EL_Width );
XtSetArg( args[6], XmNheight,              EL_Height );
XtSetValues( frame, args, 7 );

XtSetArg( args[0], XmNwidth, FM_Width );
XtSetValues( form, args, 1 );

/*
 * Setup defaults of the Work Area.
 */

XtSetArg( args[0], XmNeditable,            False );
XtSetArg( args[1], XmNsensitive,           False );
XtSetArg( args[2], XmNrightAttachment,      XmATTACH_POSITION );
XtSetArg( args[3], XmNrightPosition,        99 );
XtSetArg( args[4], XmNbottomAttachment,     XmATTACH_POSITION );
XtSetArg( args[5], XmNbottomPosition,       99 );
XtSetArg( args[6], XmNheight,              WA_Height );
XtSetArg( args[7], XmNleftOffset,           5 );
XtSetArg( args[8], XmNtopOffset,           5 );
XtSetValues( XtParent(txt_worka), args, 9 );

load_font( WA_Font, &Fnt_List );
XtSetArg( args[0], XmNfontList, Fnt_List );
XtSetValues( txt_worka, args, 1 );

XtSetArg( args[0], XmNmenuHelpWidget, btn_help );
XtSetValues( mnb_main, args, 1 );

/*
 * Setup defaults for Font selection popup.
 */

set_btn_defaults( btn_fn_ok,  DEFAULT );
set_btn_defaults( btn_fn_cnl, FALSE );
set_btn_defaults( btn_fn_hlp, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_fn_ok );
XtSetValues( dlg_font, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_font), args, 1 );

/*
 * Setup defaults for Group/Comp selection popup.
 */

set_btn_defaults( btn_group,  DEFAULT );
set_btn_defaults( btn_comp,   FALSE );
set_btn_defaults( btn_gcc,    FALSE );
set_btn_defaults( btn_gch,    FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_group );
XtSetValues( dlg_gc, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_gc), args, 1 );

/*
 * Setup defaults for get_string popup.
 */

set_btn_defaults( btn_gok,    DEFAULT );
set_btn_defaults( btn_gcnc1,  FALSE );
set_btn_defaults( btn_ghelp,  FALSE );

XtSetArg( args[0], XmNautoShowCursorPosition, TRUE );
XtSetArg( args[1], XmNcursorPositionVisible,  TRUE );
XtSetArg( args[2], XmNcursorPosition,        1 );
XtSetArg( args[3], XmNblinkRate,              0 );
```

```
XtSetValues( txt_gstr_reply, args, 4 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_gstr), args, 1 );

/*
 * Setup defaults for ask() popup.
 */

set_btn_defaults( btn_ask_y, DEFAULT );
set_btn_defaults( btn_ask_n, FALSE );
set_btn_defaults( btn_ask_h, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_ask_y );
XtSetValues( dlg_ask, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_ask), args, 1 );

/*
 * Setup defaults for user_ack() popup.
 */

set_btn_defaults( btn_ack_y, DEFAULT );
set_btn_defaults( btn_ack_h, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_ack_y );
XtSetValues( dlg_ack, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_ack), args, 1 );

/*
 * Setup defaults for display_file() popup.
 */

set_btn_defaults( btn_ok, DEFAULT );
set_btn_defaults( btn_fhelp, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_ok );
XtSetValues( dlg_file, args, 1 );

XtSetArg( args[0], XmNeditable, False );
XtSetArg( args[1], XmNsensitive, False );
XtSetValues( txt_file, args, 2 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_file), args, 1 );

/*
 * Setup defaults for archive() popup.
 */

set_btn_defaults( btn_read, DEFAULT );
set_btn_defaults( btn_list, FALSE );
set_btn_defaults( btn_write, FALSE );
set_btn_defaults( btn_cancel, FALSE );
set_btn_defaults( btn_ahelp, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_read );
XtSetValues( dlg_arch, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_arch), args, 1 );

/*
 * Setup defaults for help() popup.
 */

set_btn_defaults( btn_hlp_c, DEFAULT );
set_btn_defaults( btn_hlp_h, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_hlp_c );
XtSetValues( dlg_help, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_help), args, 1 );
```

```
XtSetArg( args[0], XmNmwmInputMode, MWM_INPUT_MODELESS );
XtSetValues( XtParent(dlg_help), args, 1 );

/*
 * Setup defaults for get_type() popup.
 */

set_btn_defaults( btn_short, FALSE );
set_btn_defaults( btn_int, FALSE );
set_btn_defaults( btn_float, FALSE );
set_btn_defaults( btn_double, FALSE );
set_btn_defaults( btn_char, FALSE );
set_btn_defaults( btn_tcancel, DEFAULT );
set_btn_defaults( btn_thelp, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_tcancel );
XtSetValues( dlg_type, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_type), args, 1 );

/*
 * Setup defaults for the help text popup.
 */

set_btn_defaults( btn_help_txt, DEFAULT );

XtSetArg( args[0], XmNdefaultButton, btn_help_txt );
XtSetValues( dlg_help_txt, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_help_txt), args, 1 );

strcpy( Help_Txt_Str, "\nThis popup allows the user to view the help text" );
strcat( Help_Txt_Str, "\nfor the Comp Builder. The user may use the scroll" );
strcat( Help_Txt_Str, "\nbars to scroll through the help text. Select close" );
strcat( Help_Txt_Str, "\nto close this window and return to the help text.\n" );

strcpy( List_Txt_Str, "\nThis popup allows the user to view listings of" );
strcat( List_Txt_Str, "\nvarious types of Comp Builder information. The" );
strcat( List_Txt_Str, "\nuser may use the scroll bars to scroll though" );
strcat( List_Txt_Str, "\nthe text.\n" );

/*
 * Setup defaults for the hello popup.
 */

set_btn_defaults( btn_hello, DEFAULT );

XtSetArg( args[0], XmNdefaultButton, btn_hello );
XtSetValues( dlg_hello, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_hello), args, 1 );

hello_str = (char *) malloc( sizeof(char) * 630 );

strcat( hello_str, "A natural language tool for\n" );
strcat( hello_str, "building real-time algorithms.\n\n\n" );
strcat( hello_str, "Designed & Built by:");
strcat( hello_str, "-----");
strcat( hello_str, "Troy Heindel, Terri Murphy");
strcat( hello_str, "Mission Operations Directorate");
strcat( hello_str, "Johnson Space Center");
strcat( hello_str, "NASA");
strcat( hello_str, "X Windows Version by:");
strcat( hello_str, "-----");
strcat( hello_str, "Timothy Barton");
strcat( hello_str, "Software Engineering Section");
strcat( hello_str, "Automation and Data Systems");
strcat( hello_str, "Southwest Research Institute");

XtSetArg( args[0], XmNlabelString, XmStringLtoRCreate( hello_str, XmSTRING_DEFAULT_CHARSET ); );
XtSetValues( lbl_hello, args, 1 );

load_font( "9x15", &Fnt_List );
XtSetArg( args[0], XmNfontList, Fnt_List );
XtSetArg( args[1], XmNlabelString, XmStringLtoRCreate(
    "CODE - Computation Development Environment", XmSTRING_DEFAULT_CHARSET ); );
XtSetValues( lbl_hello_t, args, 2 );

free( hello_str );

/*
```



```
/* Setup defaults for the device selection popup.
*/

set_btn_defaults( btn_dev_ok,  DEFAULT );
set_btn_defaults( btn_dev_cnl, FALSE );
set_btn_defaults( btn_dev_hlp, FALSE );

XtSetArg( args[0], XmNdefaultButton, btn_dev_ok );
XtSetValues( dlg_device, args, 1 );

XtSetArg( args[0], XmNtitle, " " );
XtSetValues( XtParent(dlg_device), args, 1 );

/*
 * Setup color defaults.
*/

insensitive_color = name_to_pixel( ELIColor );

XtSetArg( args[0], XmNbackground, (XtArgVal) NULL );
XtGetValues( txt_worka, args, 1 );

sensitive_color = (Pixel) args[0].value;

/*
 * Set sizes of various popups and forms.
*/

if ( SetNum == 1 )
{
    XtSetArg( args[0], XmNwidth, 425 );
    XtSetArg( args[1], XmNheight, 250 );
    XtSetValues( frm_device, args, 2 );

    XtSetArg( args[0], XmNwidth, 625 );
    XtSetArg( args[1], XmNheight, 275 );
    XtSetValues( frm_hello, args, 2 );

    XtSetArg( args[0], XmNwidth, 425 );
    XtSetArg( args[1], XmNheight, 175 );
    XtSetValues( frm_help_txt, args, 2 );

    XtSetArg( args[0], XmNwidth, 525 );
    XtSetArg( args[1], XmNheight, 100 );
    XtSetValues( frm_type, args, 2 );

    XtSetArg( args[0], XmNwidth, 325 );
    XtSetArg( args[1], XmNheight, 100 );
    XtSetValues( frm_help, args, 2 );

    XtSetArg( args[0], XmNheight, 125 );
    XtSetValues( frm_arch, args, 1 );

    XtSetArg( args[0], XmNheight, 350 );
    XtSetValues( frm_file, args, 1 );

    XtSetArg( args[0], XmNheight, 100 );
    XtSetValues( frm_ask, args, 1 );

    XtSetArg( args[0], XmNwidth, 636 );
    XtSetArg( args[1], XmNheight, 400 );
    XtSetValues( frm_gname, args, 2 );

    XtSetArg( args[0], XmNwidth, 400 );
    XtSetArg( args[1], XmNheight, 100 );
    XtSetValues( frm_gc, args, 2 );

    XtSetArg( args[0], XmNwidth, 525 );
    XtSetArg( args[1], XmNheight, 250 );
    XtSetValues( frm_font, args, 2 );
}
else
{
    XtSetArg( args[0], XmNwidth, 500 );
    XtSetArg( args[1], XmNheight, 300 );
    XtSetValues( frm_device, args, 2 );

    XtSetArg( args[0], XmNwidth, 850 );
```

```
XtSetArg( args[1], XmNheight, 350 );
XtSetValues( frm_hello, args, 2 );

XtSetArg( args[0], XmNwidth, 525 );
XtSetArg( args[1], XmNheight, 200 );
XtSetValues( frm_help_txt, args, 2 );

XtSetArg( args[0], XmNwidth, 525 );
XtSetArg( args[1], XmNheight, 100 );
XtSetValues( frm_type, args, 2 );

XtSetArg( args[0], XmNwidth, 525 );
XtSetArg( args[1], XmNheight, 150 );
XtSetValues( frm_help, args, 2 );

XtSetArg( args[0], XmNheight, 125 );
XtSetValues( frm_arch, args, 1 );

XtSetArg( args[0], XmNheight, 500 );
XtSetValues( frm_file, args, 1 );

XtSetArg( args[0], XmNheight, 100 );
XtSetValues( frm_ask, args, 1 );

XtSetArg( args[0], XmNwidth, 936 );
XtSetArg( args[1], XmNheight, 500 );
XtSetValues( frm_gname, args, 2 );

XtSetArg( args[0], XmNwidth, 650 );
XtSetArg( args[1], XmNheight, 150 );
XtSetValues( frm_gc, args, 2 );

XtSetArg( args[0], XmNwidth, 625 );
XtSetArg( args[1], XmNheight, 250 );
XtSetValues( frm_font, args, 2 );
}
```

90/06/07
14:19:46

install.c

1

INSTALL

Purpose: This routine does all the necessary stuff to move a comp built using CODE into a group of comps in the INCO Algorithm Manager.

Designer: Terri Murphy & Troy Heindel

Programmer: Terri Murphy & Troy Heindel

Date: 6/1/87

Version: 1.0

Project: CODE (Comp Development Environment)

Revised by:

Reasons for Revision:

External Interfaces

openFile() -- generic file opening routine.
save_CompInf0() -- save new comp dispositions.
inform() -- display text in the message window
ask() -- inform(), then prompt for reply
read_vars() -- read variables for a "complete" comp
variable_exist() -- see if the variable is already in the group vars list
write_comp() -- appends "complete" comps .c to group.c
config_mgmt() -- verify proper type of signals and msids
compile_group -- Compiles the group.c file
*****/

```
#include <stdio.h>
#include <string.h>
#include <sys/file.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "hisde.h"
#include "code.h"
```

FILE *group_file, /* Pointer to the file that the <GroupName>.c code will go in */
report; / Pointer to a file for standard errors */

int num_group_msids, /* The number of msids that belong to this group */
numSigsTBDef; /* the number of signals that need to be added to the signal table */

struct sig_tbl_struct sigsTBDef[MAX_NUM_SIGNALS]; /* The array of signals that need to be added
to the signal table */

```
install()
{
    FILE      *ptr1,          /* File pointer to the complete group report */
              *sig_tbl_ptr;   /* File pointer to the signal table */

    int        i,             /* Simple index */
              status,         /* return code for function calls */
              installed[MAX_GROUPS], /* Contains the index's of all comps that we
                                      are attempting to install */
              comps_installed, /* The number of comps we are attempting to install */
              group_file_name[PATH_LEN], /* Full path name of the group file */
              comp_var_count, /* The number of variable that belong to the comp that is currently
                                      being processed (Passed to translate). */
              numCompsError, /* The number of comps with disposition == ERROR */
              numCompsComplete, /* The number of comps with disposition == COMPLETE */
              numCompsIncomplete, /* The number of comps with disposition == INCOMPLETE */
              numCompsInstalled, /* The number of comps with disposition == INSTALLED */
              compileStatus; /* The status returned from compile */

    char        message[180], /* Holds messages, used with ask(),inform() */
              groupExe[80], /* The name of the group executable file */
              groupRpt[80]; /* The name of the group report */

    struct var_struct this_comps_vars[MAX_COMP_VARS]; /* The array of variables for the comp that
                                                         is currently being processed. (Passed
                                                         to translate. */

    /*
     * If there's a group executable out there, remove it and
     * change the groups disposition to complete.
     */
    sprintf(groupExe, "%s/%s", AMGroups, GroupName);
    sprintf(message, "rm -f %s 2>>/tmp/code.err", groupExe);
    if(system(message) != OK)
        user_ack(" Unable to remove old group executable ", HELP_U_ACK);
    GroupInfo[GroupNumber].disposition = COMPLETE;

    /*
     * Initially there are no group msids, installed comps
     * or signals that need defining.
     */
    num_group_msids = comps_installed = 0;
    numCompsInstalled = numCompsComplete = numCompsIncomplete = numCompsError = 0;
    numSigsTDef = 0;

    /*
     * Open a file for installation report.
     */
    sprintf(group_file_name, "%s/%s/install.rpt", CodeGroups, GroupName);
    if (!(report = openFile(group_file_name, "w", "install")))
        return (ERROR);
    fprintf(report, "\nInstallation report for %s:\n\n", GroupName);

    /*
     * We always have at least one group_var "QUALITY".
     */
    NumGroupVars = 1;
    strcpy(GroupVars[0].name, "QUALITY");
    strcpy(GroupVars[0].type, "short");
    strcpy(GroupVars[0].class, "msid");
    GroupVars[0].occurrence = 0;
    GroupVars[0].put_or_get = 0;
    GroupVars[0].lo1_limit = 0;
    GroupVars[0].lo2_limit = 0;
    GroupVars[0].hi1_limit = 0;
    GroupVars[0].hi2_limit = 0;
    strcpy(GroupVars[0].nomenclature, "/* QUALITY */");

    for(i=0; i<NumOfComps; i++)
    {
        /*
         * Get variables for comps and translate comps that
         * satisfy either of the following conditions:
         * 1. All disposition INSTALLED
         * 2. All disposition COMPLETE.
         */
    }
```

```
if (CompInfo[i].disposition == INSTALLED || CompInfo[i].disposition == COMPLETE)
{
    sprintf(message, "Installing comp : %s", CompInfo[i].name);
#ifdef HISDE
    h_message(MSG_APPLICATION,message);
#endif
    fprintf(report, "\n%s:\n", CompInfo[i].name);
    status = read_vars(CompInfo[i].name,this_comps_vars, &comp_var_count);
    if (status == ERROR)
    {
        fprintf(report, "\tVariable File Error - Notify Developers\n");
        CompInfo[i].disposition = ERROR;
    }
    status = translate(CompInfo[i].name,this_comps_vars, comp_var_count);
    if(status == ERROR)
    {
        fprintf (report, "\tTranslation Error - Notify Developers\n");
        CompInfo[i].disposition = ERROR;
    }
    else
    {
        fprintf (report, "\tTranslation Complete\n");
        installed[comps_installed++] = i;
    }
}

/*
 * If there are no comps_installed don't bother building the
 * <group>.c
 */
if(comps_installed > 0)
{
    /*
     * Let's open a file for the <GroupName>.c code and put
     * the header information in it.
     */
    sprintf (group_file_name, "%s/%s/%s.c", CodeGroups, GroupName, GroupName);
    if (!(group_file = fopen (group_file_name, "w","install")))
    {
        fprintf(report, "Install: Unable to open %s.\n", group_file);
        fclose(report);
        return (ERROR);
    }

    /*
     * Write our '#include's.
     */
    fprintf(group_file, "#include <rti.h>\n");
    fprintf(group_file, "#include <math.h>\n");
    fprintf(group_file, "#include <stdio.h>\n");
    fprintf(group_file, "#include <values.h>\n");
    fprintf(group_file, "#include <nf.h>\n");

    /*
     * Write our '#define's.
     */
    fprintf(group_file, "#define MSID_CNT %d\n", num_group_msids+1);
    fprintf(group_file, "#define COMP_CNT %d\n", comps_installed);
    fprintf(group_file, "#define GROUP_DAT %04d%s/%s.dat\042\n", AMSupport, GroupName);
    fprintf(group_file, "#define MAX_SIGNAL_LENGTH 80\n");
    fprintf(group_file, "#define MAX_NAME_LEN 11\n");

    /*
     * Define our global variables.
     */
    fprintf(group_file, "int *ind_ptr;\nint *rate_ptr;\nint *nf_ptr;\nint nf_index;\n");
    fprintf(group_file, "int iteration_ctr = 0;\nint nf[COMP_CNT];\n");
    fprintf(group_file, "short value[MSID_CNT];\nchar status[MSID_CNT];\n");

    /*
     * Loop through the group variable list.
     */
    fprintf(group_file, "/* Signal List */\n");
    for(i=0;i<NumGroupVars;i++)
    {
        /*
         * If we find a signal then define it.
         */
    }
}
```

```

/*
if(strcmp(GroupVars[i].class, "signal") == 0)
{
    fprintf(group_file, "%s %s", GroupVars[i].type, GroupVars[i].name);
    /*
     * If the signal is a char string, define the string
     * length to be MAX_SIGNAL_LENGTH long.
     */
    if (strcmp (GroupVars[i].type, "char") == 0)
        fprintf (group_file, "[MAX_SIGNAL_LENGTH];\n");
    else fprintf (group_file, ";\n");
}
}

fprintf(group_file, "/* End Signal List */\n\n");

/*
 * Define the main!!!!
 */
fprintf(group_file, "main(argc, argv)\n int argc;\n char *argv[];\n\n");

/*
 * Loop through the group variable list.
 */
fprintf(group_file, "\tstatic char *msid_list[] = {\n";
for(i=0; i<NumGroupVars; i++)
{
    /*
     * If we find an MSID then define it.
     */
    if(strcmp(GroupVars[i].class, "msid") == 0)
        fprintf(group_file, "\t\t\t\t\t042%s\042,\n", GroupVars[i].name);
}
fprintf(group_file, "\t\t\t\t\t042\042\n\t\t\t\t\t);\n\n");

/*
 * Declare our remaining locals.
 */
fprintf(group_file, "\tFILE *in;\n\n");
fprintf(group_file, "\tint indx_array[MSID_CNT];\n\tint i=0;\n\tint disposition;\n");
fprintf(group_file, "\tint ind_array[COMP_CNT];\n\tint rate_array[COMP_CNT];\n\n");
fprintf(group_file, "\tchar purpose[80];\n");
fprintf(group_file, "\tchar CompName[MAX_NAME_LEN];\n\n");

/*
 * Declare external fault message routines.
 */
fprintf(group_file, "\tvoid fltmsg_init();\n\tint fltmsg_issue();\n\n");

/*
 * Open the .dat file and read in the CompName
 * noise filter, rate, on/off, disposition and purpose.
 */
fprintf(group_file, "\t/\n*****\n");
fprintf(group_file, "\tOpen <GroupName>.dat file and read in the\n");
fprintf(group_file, "\tcomp name (which isn't used), noise filter, \n");
fprintf(group_file, "\trate, on/off, disposition and purpose \n");
fprintf(group_file, "\tfor all comps in the group.\n");
fprintf(group_file, "\t*****\n");
fprintf(group_file, "\tif ((in = fopen(GROUP_DAT, \042r\042)) == NULL)\n");
fprintf(group_file, "\t{\n\t\t\t\t\tfltmsg_issue(\042Unable to open %s file\\n\042, GROUP_DAT);\n");
fprintf(group_file, "\t\t\t\t\texit(-1);\n\t}\n");
fprintf(group_file, "\tfscanf(in, \042%*[^\\n]\042);\n");
fprintf(group_file, "\twhile(fscanf(in, \042%*[^\\n]\042, CompName, \n");
fprintf(group_file, "\t\t\t\t\t&nf[i], \n");
fprintf(group_file, "\t\t\t\t\t&rate_array[i], \n");
fprintf(group_file, "\t\t\t\t\t&ind_array[i], \n");
fprintf(group_file, "\t\t\t\t\t&disposition, \n");
fprintf(group_file, "\t\t\t\t\tpurpose) != EOF) i++; \n");
fprintf(group_file, "\tfclose(in);\n\n");

/*
 * Attach to shared memory for telemetry and fault
 * messages, initialize fault message, and get
 * the array of msid indexes.
 */
fprintf(group_file, "\ttattach_shmem(STD);\n");
fprintf(group_file, "\tfltmsg_init();\n");
fprintf(group_file, "\tgetmsid_indx_lst(msid_list, indx_array);\n\n");

```

install.c

5

```

/*
 * Now for the loop...
 */
fprintf(group_file, "\twhile(++iteration_ctr)\n\t{\n");
fprintf(group_file, "\t\tdo\n\t\t{\n\t\t\tgetmsid_off_data_1st(indx_array; value, status);\n\t\t} while (
value[0] == 0);\n\n");
fprintf(group_file, "\t\tind_ptr = ind_array;\n");
fprintf(group_file, "\t\ttrate_ptr = rate_array;\n");
fprintf(group_file, "\t\ttnf_index = 0;\n\n");

for(i=0;i<comps_installed;i++)
{
    fprintf(group_file, "\t\tif((*ind_ptr++ == 1) && (iteration_ctr %% *rate_ptr) == 0)\n");
    fprintf(group_file, "\t\t\t{\n\t\t\t\tCompInfo[installed[i]].name);
    fprintf(group_file, "\t\t\t\ttnf_index;\n\t\t\t\ttrate_ptr;\n\n");
}

fprintf(group_file, "\t}\n\n");

/*****
Here's where we write the comp's C code that's
generated by translate.
*****/
for(i=0;i<comps_installed;i++)
{
    if(write_comp(CompInfo[installed[i]].name) == ERROR)
    {
        fprintf(report, "Install: Unable to write %s's 'C' version.\n", CompInfo[installed[i]].name);
        fprintf(group_file, "%s()\n\n\treturn;\n);\n\n", CompInfo[installed[i]].name);
        CompInfo[installed[i]].disposition = ERROR;
    }
    else CompInfo[installed[i]].disposition = INSTALLED;
}

fclose(group_file);

/*****
Compile and save the group variables and the updated dispositions.
*****/
compileStatus = compile_group();

if(compileStatus == ERROR)
{
    /*
     * If we couldn't compile - then we need to change the comp dispositions
     * back to COMPLETE, and the group disposition to ERROR.
     */
    for (i=0;i<NumOfComps;i++)
    {
        if(CompInfo[i].disposition == INSTALLED)
            CompInfo[i].disposition = COMPLETE;
    }
}

/*****
Report on the new disposition of the comps.
*****/
fprintf(report, "\n\nComp Name\tDisposition\n");
fprintf(report, "_____\n");
for(i=0;i<NumOfComps;i++)
{
    switch(CompInfo[i].disposition)
    {
        case ERROR: fprintf(report,"%s\t\tERROR\n", CompInfo[i].name);
                    numCompsError++;
                    break;

        case COMPLETE: fprintf(report,"%s\t\tCOMPLETE\n",CompInfo[i].name);
                        numCompsComplete++;
                        break;

        case INCOMPLETE:fprintf(report,"%s\t\tINCOMPLETE\n", CompInfo[i].name);
                            numCompsIncomplete++;
                            break;
    }
}

```

```

        case INSTALLED: fprintf(report, "%s\t\tINSTALLED\n", CompInfo[i].name);
                        numCompsInstalled++;
                        break;

        default:      break;
    }
}
fprintf(report, "\n\nTotal Comps:\t\t\t%d\n", NumOfComps);
fprintf(report, "Installed comps:\t\t\t%d\n", numCompsInstalled);
fprintf(report, "Incomplete comps:\t\t\t%d\n", numCompsIncomplete);
fprintf(report, "Comps containing errors:\t\t\t%d\n\n", numCompsError);

if(comps_installed > 0)
{
    if(compileStatus != ERROR)
    {
        /*
         * Since the group was successfully installed, we need to add
         * the signals to the signal table and report what signals were
         * added. But before we do that let's make sure the signal
         * table exists and don't do anything if it does not.
         */
        if(access(SignalTbl, F_OK) != ERROR)
        {
            fprintf(report, "Signals defined in %s as a result of installing group %s:\n", SignalTbl, GroupName);
            if (!(sig_tbl_ptr = fopen (SignalTbl, "a")))
            {
                sprintf(message, " Warning: Could not open '%s' for writing. ", SignalTbl);
                user_ack(message, HELP_U_ACK);
            }
        }
        else
        {
            /*
             * If we're adding signals to the signal table, we need
             * to need to make sure that we start on a new line.
             */
            if (numSigsTBDef > 0)
                fprintf(report, "\n");
            /*
             * Add the new signals.
             */
            for(i=0; i<numSigsTBDef; i++)
            {
                /*
                 * Since we're adding it to the file - we better also add to
                 * our signal table structure in memory in case we install
                 * again without retrieving!!!
                 */
                strcpy(SignalTable[SignalCount].name, sigsTBDef[i].name);
                strcpy(SignalTable[SignalCount].type, sigsTBDef[i].type);
                strcpy(SignalTable[SignalCount].nomenclature, sigsTBDef[i].nomenclature);
                SignalCount++;
                /*
                 * If the type is 'c' the signal table expects
                 * the type to be 'S'
                 */
                if(sigsTBDef[i].type[0] == 'c')
                    sigsTBDef[i].type[0] = 'S';
                fprintf(sig_tbl_ptr, "%s %c %s\n", sigsTBDef[i].name, sigsTBDef[i].type[0], sigsTBDef[i].nomenclature);
                fprintf(report, "\t%s %c %s\n", sigsTBDef[i].name, sigsTBDef[i].type[0], sigsTBDef[i].nomenclature);
            }
            fclose (sig_tbl_ptr);
        }
        fprintf(report, "\n\n");
        fprintf(report, "*****\n");
        fprintf(report, "\tCONGRATULATIONS - %s has been successfully compiled.\n", GroupName);
    }
    else
    {
        fprintf(report, "\n\n");
        fprintf(report, "*****\n");
        fprintf(report, "\tUnable to compile - Please notify developers\n");
    }
}

```


90/06/07
14:19:46

install.c

7

```
    }
    else
    {
        fprintf(report, "\n\n");
        fprintf(report, "*****\n");
        fprintf(report, "\tCompilation not attempted - No comps installed.\n");
    }
    fprintf(report, "*****\n");
    fclose(report);
    save_CompInfo();

    /*****
    Combine the date, install report, and compile report into one
    master <group_name>.rpt
    *****/
    system("date '+DATE: %h %d, %Y%TIME: %H:%M:%S%' >/tmp/date 2>>/tmp/code.err");
    sprintf(message, "cat /tmp/date %s/%s/install.rpt %s/%s/compile.rpt >%s/%s/%s.rpt 2>>/tmp/code.err", CodeGroups,
    ps, GroupName, CodeGroups, GroupName, CodeGroups, GroupName, GroupName);
    if(system(message) != 0)
    {
        user_ack(" Install: Unable to create install/compile report. ", HELP_U_ACK);
    }

    /*****
    Write some blank lines to the end of the report,
    so we can see error messages when we "more" the report.
    *****/
    sprintf(groupRpt, "%s/%s/%s.rpt", CodeGroups, GroupName, GroupName);
    if (!(ptr1 = fopen (groupRpt, "a")))
    {
        user_ack(" Install: Unable to write group report. ", HELP_U_ACK);
    }
    else
    {
        for(i=0; i<60; i++)
            fprintf(ptr1, "\n");
        fclose(ptr1);
    }
    chmod(666, groupRpt);
    /*****
    Show the "MORE" prompt and select the work area
    window for the output of the system call.
    *****/
    sprintf(message, "cp %s/%s/%s.rpt /tmp/code.tmp ", CodeGroups, GroupName, GroupName );
    system( message );
    display_file( LIST, "/tmp/code.tmp" );

    /*****
    Print the report if desired.
    *****/
    sprintf(message, "Print report? (Y/N)");

    if( ask(message) )
    {
        sprintf(message, "lpr %s/%s/%s.rpt 2>>/tmp/code.err", CodeGroups, GroupName, GroupName);
        if(system(message) != 0)
        {
            user_ack("Install: Unable to create install/compile report.", HELP_U_ACK);
        }
    }

    return( OK );
}
```

```
read_vars(compName, this_comps_vars, comp_var_count)
char compName[];
struct var_struct this_comps_vars[];
int *comp_var_count;
{
    FILE *var_file;

    char variable_file[PATH_LEN];

    int index; /* The index returned from variable_exists if it exists */

    sprintf(variable_file, "%s/%s.%s.v", CodeGroups, GroupName, compName);
    if (!(var_file = openFile(variable_file, "r", "install")))
    {
        fprintf(report, "Cannot find variables for %s.\n", variable_file);
        return (ERROR);
    }

    /*****
     First we will discard the comment
     that is the 1st line of all variable files.
     *****/
    fscanf(var_file, "%*[^\n]");
    /*****/
    Now let's read the variable info for this
    comp until we reach the end of the file.
    *****/
    *comp_var_count = 0;
    while(fscanf(var_file, "%s %s %s %d %d %f %f %f %f %[^\n]",
                this_comps_vars[*comp_var_count].name,
                this_comps_vars[*comp_var_count].type,
                this_comps_vars[*comp_var_count].class,
                &this_comps_vars[*comp_var_count].occurrence,
                &this_comps_vars[*comp_var_count].put_or_get,
                &this_comps_vars[*comp_var_count].lo1_limit,
                &this_comps_vars[*comp_var_count].lo2_limit,
                &this_comps_vars[*comp_var_count].hi1_limit,
                &this_comps_vars[*comp_var_count].hi2_limit,
                this_comps_vars[*comp_var_count].nomenculture) != EOF)
    {
        /*****/
        Let's do some CM if it's an MSID or a signal.
        *****/
        if(strcmp(this_comps_vars[*comp_var_count].class, "msid") == 0 ||
            strcmp(this_comps_vars[*comp_var_count].class, "signal") == 0)
        {
            config_mgmt(this_comps_vars[*comp_var_count].name,
                        this_comps_vars[*comp_var_count].class,
                        this_comps_vars[*comp_var_count].type,
                        this_comps_vars[*comp_var_count].nomenculture);
        }

        /*****/
        First we need to see if the variable already
        exists in the group_var structure. If it doesn't
        exist we will increment the NumGroupVars and
        copy the *comp_var into the GroupVars list.
        If it existed we increment the occurrence
        field of the group_var
        *****/
        if ((index = variable_exists(this_comps_vars[*comp_var_count].name,
                                    this_comps_vars[*comp_var_count].type,
                                    this_comps_vars[*comp_var_count].class)) == ERROR)
        {
            GroupVars[NumGroupVars] = this_comps_vars[*comp_var_count];
            /*****/
            Let's keep track of how many msids we have
            because we need to "#define MSID_CNT" in
            the <group>.c file.
            *****/
            if(strcmp(GroupVars[NumGroupVars].class, "msid") == 0)
                num_group_msids++;

            NumGroupVars++;
        }
        else GroupVars[index].occurrence++;
    }
}
```

90/06/07
14:19:46

install.c

9

```
    (*comp_var_count)++;  
}  
fclose (var_file);  
return(0);  
}
```

```
variable_exists(name, type, class)
char name[];
char type[];
char class[];
{
    int i;

    for(i=0; i<NumGroupVars;i++)
        if (strcmp(GroupVars[i].name, name) == 0)
        {
            return(i);
        }
    return(ERROR);
}
```

```

save_CompInfo()
{
    FILE    *CompInfo_file;

    char CompInfo_name[PATH_LEN];

    int i; /* Simple index */

    sprintf (CompInfo_name, "%s/%s.dat", AMSupport, GroupName);
    /*****
    Open group info file and write variables
    *****/
    if (!(CompInfo_file = openFile (CompInfo_name,"w","install")))
        return (ERROR);

    /*****
    Write all of the group info for the group with header.
    *****/
    fprintf (CompInfo_file, "#name_name noise_filter rate on_off disposition\n");
    for (i=0; i < NumOfComps; i++)
    {
        fprintf (CompInfo_file, "%s %d %d %d %d %s\n", CompInfo[i].name,
                    CompInfo[i].noise_filter,
                    CompInfo[i].rate,
                    CompInfo[i].on_off,
                    CompInfo[i].disposition,
                    CompInfo[i].purpose);
    }

    fclose(CompInfo_file);
}

```

```

write_comp(name)
char name[];
{
    FILE *ptr;

    char the_file[PATH_LEN], ch;

    sprintf(the_file, "%s/%s/%s.c", CodeGroups, GroupName, name);

    if (!(ptr = openFile(the_file, "r", "install")))
    {
        fprintf(report, "write_comp: unable to open %s.\n", the_file);
        return (ERROR);
    }

    while ( (ch = fgetc (ptr)) != EOF)
    {
        fputc (ch, group_file);
    }

    fputc ('\n', group_file);
    fputc ('\n', group_file);
    fclose(ptr);

    return( OK );
}

```

```
config_mgmt(name, class, type, nomenclature)
char name[];
char class[];
char type[];
char nomenclature[];
{
    int i;
    char compare_type[TYPE_LEN]; /* the corrected type from the signal table 'S' -> 'c' */

    /*****
    If it's a signal make sure is in the SignalTable list
    *****/
    if(strcmp(class, "signal") == 0)
    {
        for (i=0; i < SignalCount; i++)
        {
            if (strcmp (name, SignalTable[i].name) == 0)
            {
                if (SignalTable[i].type[0] == 'S')
                    strcpy(compare_type, "c");
                else strcpy (compare_type, SignalTable[i].type);
                /*****
                It's in the signal table - verify
                that the type is correct.
                *****/
                if(type[0] != compare_type[0])
                {
                    fprintf(report, "\t%s type is not consistent with signal table.\n", name);
                    return(ERROR);
                }
                else return(OK);
            }
        }
        /*****
        It's not defined in the signal table.
        See if it's already in our array of
        signals that need to be defined.
        *****/
        for (i=0; i<numSigsTBDef; i++)
        {
            if(strcmp(sigsTBDef[i].name, name) == 0)
            {
                /*****
                If it's already in our list of signals to be defined
                make sure it has the same type and nomenclature.
                *****/
                if(strcmp(sigsTBDef[i].type, type)==0 &&
                    strcmp(sigsTBDef[i].nomenclature, nomenclature)==0)
                {
                    return(OK);
                }
                else
                {
                    fprintf(report, "\t%s type is not consistent with previous definition.\n", name);
                    return(ERROR);
                }
            }
        }
        /*****
        It's not in the list to be defined so put it there.
        *****/
        strcpy(sigsTBDef[numSigsTBDef].name, name);
        strcpy(sigsTBDef[numSigsTBDef].type, type);
        strcpy(sigsTBDef[numSigsTBDef].nomenclature, nomenclature);
        numSigsTBDef++;
        return(OK);
    }
    else if (strcmp(class, "msid") == 0)
    {
        for (i=0; i < MSIDCount; i++)
        {
            if (strcmp (name, MSIDTable[i].name) == 0)
            {
                if(strcmp(type, MSIDTable[i].type) != 0)
                {
                    fprintf(report, "\t%s type is not consistent with tag_msid_table.\n", name);
                    return(ERROR);
                }
            }
        }
    }
}
```

80/06/07
14:19:46

install.c

14

```
    }  
    else return(OK);  
}  
}  
fprintf(report, "\\t%s is not defined in the tag_msid_table.\\n", name);  
return(ERROR);  
}  
  
return( OK );  
}
```



```
/*  
    compile_group
```

Purpose: Compile_group does a system call on cc, compiling the
the C file in order to create an executable. It
compiles with the correct libraries and include files.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 8/18/87

Version: 1.0

Project: CODE (Comp Development Environment)

```
*****/  
compile_group ()  
{  
    char    sys_cmd[500],          /* String to hold system command. */  
            compileRpt[PATH_LEN], /* String to hold file name of compile report */  
            libPath[PATH_LEN];     /* string to build path to the libraries */  
  
    int     rc;                    /* Return code for system command. */  
  
    /*  
     * Compile the group and redirect errors to the installation report  
     * <GroupName>.rpt.  
     */  
#ifdef HISDE  
    h_message(MSG_APPLICATION, "Please wait, compiling...");  
#endif  
    sprintf(compileRpt, "%s/%s/compile.rpt", CodeGroups, GroupName);  
    sprintf(libPath, "%s/rtds/lib", RTDS);  
    sprintf(sys_cmd, "cc -o %s/%s %s/%s/%s.c -I$RTDS/rtds/include %s/libuserfuncs.a %s/libbnf.a %s/libfltmsg.a %s  
/librti.a %s/libiesputil.a 1>%s 2>%s", AMGroups, GroupName, CodeGroups, GroupName, GroupName, libPath, libPath, l  
ibPath, libPath, libPath, compileRpt, compileRpt);  
    rc = system (sys_cmd);  
  
    /*  
     * If there were compilation errors return an ERROR.  
     */  
    if(rc != 0)  
    {  
        GroupInfo[GroupNumber].disposition = ERROR;  
        /*  
         * Put the group names into the GroupInfo file  
         */  
        writeGroupNames();  
        return(ERROR);  
    }  
    else  
    {  
        GroupInfo[GroupNumber].disposition = INSTALLED;  
        /*  
         * Put the group names into the GroupNames file  
         */  
        writeGroupNames();  
        return(OK);  
    }  
}
```

90/06/07
14:19:57

menu.c

1

```
/*-----*/
*
* FILE NAME:    menu.c
*
*
* FILE FUNCTION:
*
*   Contains the routines which set the state of the Comp element (if, then, etc)
*   push buttons.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* FILE MODULES:
*
*   all_invalid()  - sets all Comp element push buttons to invalid state
*   all_valid()    - sets all Comp element push buttons to active state
*   color_valid()  - sets the currently valid Comp element push buttons to active
*
*-----*/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include "code.h"
#include "widgets.h"
```

```

/*****<----->*****/
*
* MODULE NAME:  all_invalid( void )
*
*
* MODULE FUNCTION:
*
*   Sets all Comp element (if, then, MSID, etc) tokens to an invalid/inactive state.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                      Data Systems Department
*                      Automation and Data Systems Division
*                      Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
/*****<----->*****/

void all_invalid()
{
    int i;
    Arg args[2];

    /*
     * Set all tokens invalid except for the non-comp element tokens which are
     * always valid.
     */

    XtSetArg( args[0], XmNsensitive, (XtArgVal) FALSE );
    XtSetArg( args[1], XmNbackground, (XtArgVal) insensitive_color );

    for ( i=IF; i<=NUMBER; i++ )
        XtSetValues( w[i], args, 2 );
    for ( i=EQ; i<=STOP; i++ )
        XtSetValues( w[i], args, 2 );
    for ( i=ADD; i<=ATAN; i++ )
        XtSetValues( w[i], args, 2 );
}

```

90/06/07
14:19:57

menu.c

3

```

/*****<----->*****/
*
* MODULE NAME:  all_valid( void )
*
*
* MODULE FUNCTION:
*
*   Sets all Comp element (if, then, MSID, etc) tokens to a valid/active state.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Motif Release 1.0 - 90/03/16
*
*****/
```

```
void all_valid()
```

```
{
    int i;
    Arg args[2];

    XtSetArg( args[0], XmNsensitive, (XtArgVal) TRUE );
    XtSetArg( args[1], XmNbackground, (XtArgVal) sensitive_color );

    for ( i=IF; i<=NUMBER; i++ )
        XtSetValues( w[i], args, 2 );
    for ( i=EQ; i<=STOP; i++ )
        XtSetValues( w[i], args, 2 );
    for ( i=ADD; i<=ATAN; i++ )
        XtSetValues( w[i], args, 2 );
}
```

```
/*-----*/
*
* MODULE NAME: color_valid( previous, other )
*
*
* MODULE FUNCTION:
*
* Based on the currentl location in the Comp, this routine sets the valid Comp
* element push buttons to active and invalid Comp element push buttons to
* inactive.
*
*
* SPECIFICATION DOCUMENTS:
*
* /code/specs/code
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
* Timothy J. Barton - Software Engineering Section
* Data Systems Department
* Automation and Data Systems Division
* Southwest Research Institute
*
*
* REVISION HISTORY:
*
* Motif Release 1.0 - 90/03/16
*
*-----*/
```

```
void color_valid( previous, other )

    int other,
        previous;

    int i;
    Arg args[2];

    next_inputs( previous, other );

    if ( previous < 0 || previous > 50 )
        previous = 0;

    /*
     * Set all tokens invalid.
     */

    all_invalid();

    /*
     * Set the COMMENT to full intensity.
     */

    XtSetArg( args[0], XmNsensitive, (XtArgVal) TRUE );
    XtSetArg( args[1], XmNbackground, (XtArgVal) sensitive_color );

    XtSetValues( w[ COMMENT ], args, 2 );

    /*
     * Set the valid tokens to full intensity.
     */

    XtSetArg( args[0], XmNsensitive, (XtArgVal) TRUE );
    XtSetArg( args[1], XmNbackground, (XtArgVal) sensitive_color );

    for ( i=0; i < TotValPts; i++ )
        XtSetValues( w[ ValidPoints[i] ], args, 2 );
```

90/06/07
14:20:02

new.c

1

NEW

Purpose: Prompts the user for the name of a new group or comp.

Designer: Troy Heindel

Programmer: Troy Heindel

Date: 2/8/89

Version: 2.0

Project: CODE (Comp Development Environment)

*****/

#include <stdio.h>

#include <string.h>

#include <X11/Intrinsic.h>

#include <Xm/Xm.h>

#include "code.h"

90/06/07
14:20:02

new.c

2

```
new(theClass)
char theClass[]; /* Either "Comp" or "Group" */

char nameToGet[MAX_NAME_LEN], group_dir[PATH_LEN];

/*
 * Clean the mean screen in prep for the new comp.
 */
cleanSlate();

/*
 * If new Group, blank out the group name. Otherwise, if
 * new Comp, blank out the comp name.
 */

if (!strcmp(theClass, "Group"))
{
    GroupName[0] = NULL;
    CompName[0] = NULL;
}
else
    CompName[0] = NULL;

put_status();

/*
 * Get the new name from the user, if the user defaults
 * we return to calling routine with no changes.
 */

if (get_new_name(theClass, nameToGet) == DEFAULT)
    return(DEFAULT);

/*
 * Need to get a Comp
 */

if (!strcmp(theClass, "Comp"))
{
    CompNumber = NumOfComps++;
    strcpy (CompName, nameToGet);
    strcpy (CompInfo[CompNumber].name, CompName);
    CompInfo[CompNumber].disposition = INCOMPLETE;
    CompInfo[CompNumber].cycle_mode = CYCLIC;
    CompInfo[CompNumber].rate = 1;
    CompInfo[CompNumber].on_off = FALSE;
    CompInfo[CompNumber].noise_filter = 2;
    put_status();
}
else /* It was a group */
{
    GroupNumber = NumOfGroups;
    strcpy (GroupName, nameToGet);
    strcpy (GroupInfo[NumOfGroups].name, GroupName);
    /*
     * This is the only place where NumOfGroups++
     */
    GroupInfo[NumOfGroups++].disposition = INCOMPLETE;
    /*****
     * Create the necessary directory since it does not exist yet.
     *****/
    sprintf (group_dir, "%s/%s", CodeGroups, GroupName);
    mkdir (group_dir, 511);
    NumOfComps = 0;
    put_status();
}

return( OK );
```

```
get_new_name(theClass,nameToGet)
char theClass[], /* Whether it is a 'Comp' or 'Group' */
      *nameToGet; /* The name we shall set within */

char tmpStr[200];
int i, status;

do
{
    sprintf(tmpStr,"Please type the new %s name(<%d char)",theClass,MAX_NAME_LEN-2);
    status = get_string(tmpStr, nameToGet, MAX_NAME_LEN-1,FALSE);
    if(status == ABORT || nameToGet[0]!='q' || nameToGet[0]!='Q')
        return(DEFAULT);

    /*
     * File names may not have a leading zero
     */

    if (nameToGet[0] >= '0' && nameToGet[0] <= '9')
    {
        status = ERROR;
        user_ack("First character must be alphabetic", HELP_U_ACK );
    }

    /*
     * Make sure that it's not an already existing name.
     */

    else
    {
        /*
         * The comp name is called as a subroutine
         * and therefore can not have the name main.
         */
        if (!strcmp("main",nameToGet))
        {
            status = ERROR;
            user_ack("'main' is reserved; do not use it.", HELP_U_ACK);
        }
        else if (!strcmp(theClass,"Group"))
        {
            /*
             * Look for a duplicate group name
             */
            for (i=0;i<NumOfGroups;++i)
            {
                if (!strcmp (GroupInfo[i].name, nameToGet))
                {
                    status = ERROR;
                    sprintf(tmpStr, "%s already exists.", nameToGet);
                    user_ack(tmpStr, HELP_U_ACK);
                    break;
                }
            }
        }
        else /* Look through the list of comps */
        {
            /*
             * Look for a duplicate comp name
             */
            for (i=0;i<NumOfComps;++i)
            {
                if (!strcmp (CompInfo[i].name, nameToGet))
                {
                    status = ERROR;
                    sprintf(tmpStr, "\n\t%s already exists",nameToGet);
                    user_ack(tmpStr, HELP_U_ACK);
                    break;
                }
            }
        }
    }
} while (status != OK);
return OK;
```


**SPECIFICATION DRIVEN
LANGUAGE**

**SWRI PROJECT 05-2768
NASA GRANT NAG 3-339**

FINAL REPORT

June 12, 1990

**Steven W. Dellenback, Ph.D.
Timothy J. Barton**

AGENDA

-> Introduction

Specification Driven Language Research

Flight Certified Compiler

Image Processing

Conclusion

INTRODUCTION

The grant was awarded to perform basic research in the area of alternative methods to develop software programs.

Major efforts performed within the grant included:

- Investigate alternative languages currently in use (CODE, UIL, GOAL)
- Analysis and prototype implementation of CODE (or Comp Builder) in an X-Windows environment
- Investigated alternative user input techniques
- Investigated flight certified compiler concepts

AGENDA

Introduction

-> Specification Driven Language Research

Flight Certified Compiler

Image Processing

Conclusion

LANGUAGE RESEARCH

Goals:

- Provide a screen oriented programming language and/or environment for flight controller applications
- Language and/or environment should be structured so that programs are flight certifiable
- Application must retrieve mission data in a safe and efficient manner
- Language should be portable to the following environments:
 - Space Shuttle (MCC)
 - Space Station (SSCC)
 - Payloads (TSS)

LANGUAGE RESEARCH

Project Activities:

- Investigated languages used by NASA and other organizations
 - User Interface Language (UIL)
 - Ground Operations Aerospace Language (GOAL)
 - Transportable Application Environment (TAE)
- Evaluated existing NASA prototype - Comp Builder (CODE)
 - Developed BNF representation of CODE language
 - Evaluated CODE software design and implementation
- Assisted in the definition of the environment requirements
 - NASA Computation Builder and Computation Manager Level B Requirements
- Developed a BNF representation of the Mission Operations Application Language (MOAL)

LANGUAGE RESEARCH

Project Activities:

- Develop prototypes based on requirements
 - X Windows - HP Widget prototype
Interface enhancements
Implementation enhancements
 - X Windows - MOTIF prototype
Continued inteface enhancements
Implemented MOAL language constructs

AGENDA

Introduction

Specification Driven Language Research

-> Flight Certified Compiler

Image Processing

Conclusion

FLIGHT CERTIFIED COMPILER

Goal:

- To allow a user to develop "flight certified software" automatically (no human certification required).

Definition:

- Target language generation is reproducible
- Execution of the target language will cause no abnormal events on the host computer or any connected LANs (and computers)
- Program will always generate correct results

LANGUAGE ALTERNATIVES

A language designed for the flight certified compiler should, at a minimum, include the following constructs:

- Named, strongly typed variables/constants
- Assignment operators
- Loop constructs (pre- and post-conditional loops)
- Conditional constructs (single and multi-case selection)
- Modularization constructs (e.g., functions)
- Data acquisition constructs (provide a mechanism for programs to efficiently and reliably acquire LAN data)
- Input/output constructs

LANGUAGE DEFINITION

- Compilation Process
 - Lexical analysis
 - Parsing
 - Code generation
 - Binding
- Representation Forms
 - Context Free Grammars
 - Chomsky Normal Form
 - Griebach Normal Form
 - LR Grammars

LANGUAGE IMPLEMENTATION

- Flight certification will occur in two areas:
 - Language Specification
 - Language Implementation
- Implementation Alternatives:
 - Machine Code Generation
 - . Generation options
 - Direct generation
 - Intermediate language
 - . Advantages
 - . Disadvantages
 - Table-driven Execution
 - . Concept
 - . Advantages
 - . Disadvantages

VERIFICATION

- Goal
- Grammar Specification Verification
- Language Implementation Verification

AGENDA

Introduction

Specification Driven Language Research

Flight Certified Compiler

-> Image Processing

Conclusion

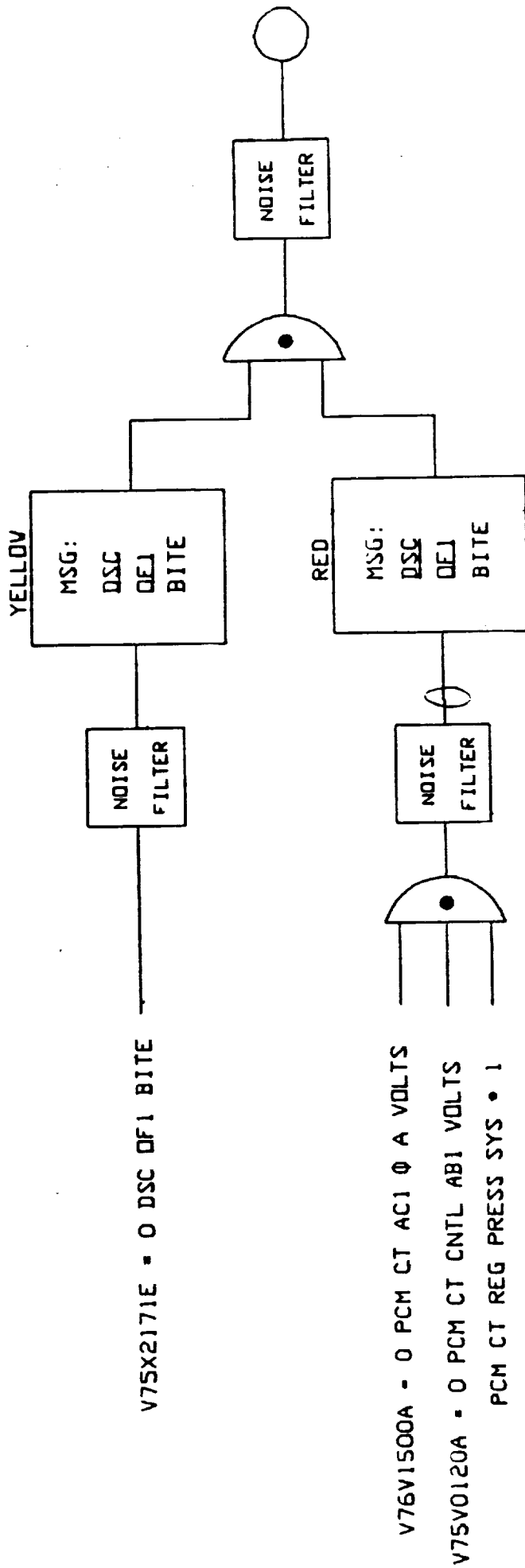
IMAGE PROCESSING

Goal:

- To determine the capabilities of image processing and to determine if an optical scanner COTS package exists which converts logic drawings into 'C' language programs.

Background:

- MCC Flight Control positions currently maintain various types of logic diagrams which are used to check the state of their system
- These logic diagrams are manually converted into Comps
- Several flight control positions manually generate test plans to ensure the logic contained in the Comps matches the logic diagram
- Does a COTS package exist which could automatically generate Comps from the diagrams?



SIGNATURE	DATE	NASA-JOHNSON SPACE CENTER		
DSGN		TITLE - DSC DEF STATUS INDICATOR		
DR C. L. H.		OWNER - EALICK/STAFFORD		
ENGR		FAULT_MSG -		
ENGR		CLASS -		
APP		DVG NO 17		
APP				
DUAL				
LTR	PCN		PAGE / - /	SHEET OF

EXAMPLE COMP

```

/*****
Creation Time: 7208:16:42:36      Author: Troy Heindel
Group Name: DSC_STATUS           Comp Name: OF1
Purpose: DSC Status Indicator (p.1-1) (Algorithm #17)
*****/
if V75X2171EC == 0
then print2 "DSC OF1 Byte"
endif
if V76V1500AC == 0 and
V76V0120AC == 0 and
V76V0120AD == 0
then print1 "DSC OF1 Fail"
endif
if V75X2171EC == 0 and
V76V1500AC == 0 and
V76V0120AC == 0 and
V76V0120AD == 0
then print1 "OF1 Status"
endif
/
```

IMAGE PROCESSING/IMAGE RECOGNITION MARKET SURVEY

COTS Package Does Not Exist Which Meets Requirements

Available COTS Software And Hardware:

- Optical scanner hardware is a rapidly expanding market
 - Similar technology to photocopiers and laser printers
 - 250 percent annual growth since 1984
- Optical Character Recognition (OCR) Software
- Computer Aided Design (CAD) Software
- CAD overlay software
- Graphics drawing software

IMAGE PROCESSING/IMAGE RECOGNITION RESEARCH

IEEE Transactions on Pattern Analysis and Machine Intelligence Identify Thousands of Research Projects

- Most address more fundamental aspects of image processing or image recognition:
 - Noise reduction and filtering
 - Edge enhancement and edge detection
 - Feature extraction
 - Raster-to-vector conversion
 - Dithering, gray scaling
- As fundamental aspects are established, various technologies are combined:
 - "A Robust Algorithm for Text String Separation From Mixed Text/Graphics Images" - Fletcher and Kasturi
 - "An Automatic Circuit Diagram Reader With Loop-Structure-Based Symbol Recognition" - Toshiba

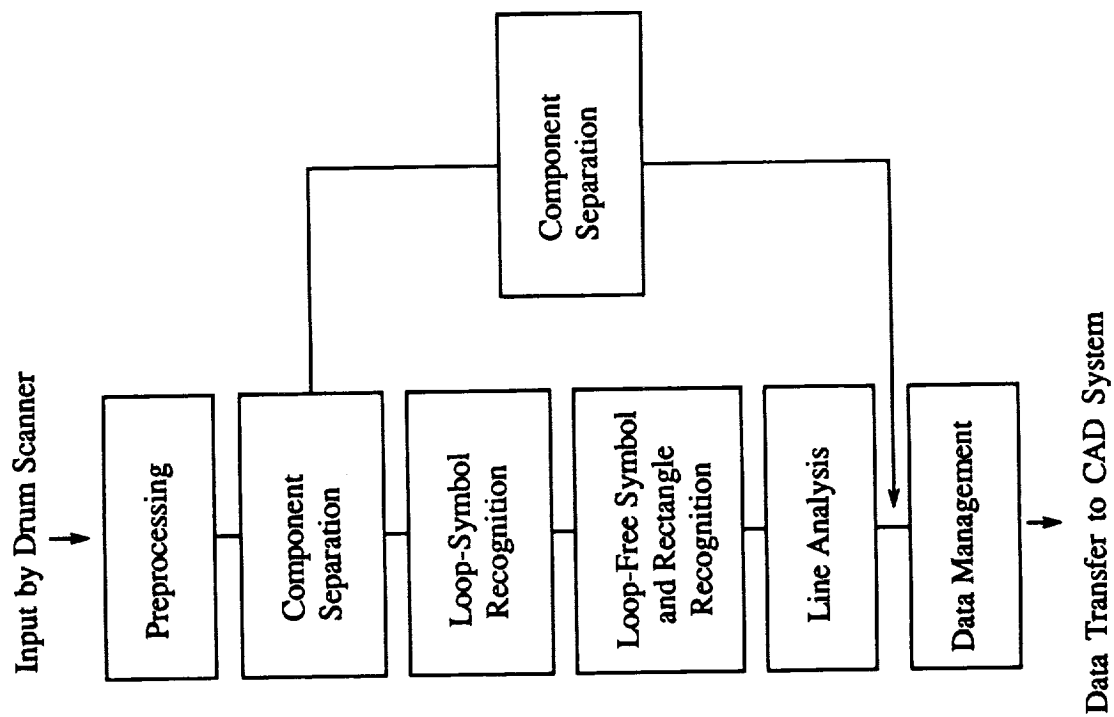


IMAGE PROCESSING/IMAGE RECOGNITION CONCLUSIONS

Conclusions:

- COTS package does not exist
- Technology is rapidly producing the required image processing/image recognition elements
- Research prototypes exist

Research Directions:

- Define drawing standard and prototype scanner/recognition process
- Computer based development environment

AGENDA

Introduction

Specification Driven Language Research

Flight Certified Compiler

Image Processing

-> Conclusion

LUSIONS

- Alternative languages are valuable and useful tools
- Existing languages (e.g. CODE, GOAL) have demonstrated the benefits of utilizing alternative languages
- X-Windows provides a good environment to develop a user interface for program development application
- Research efforts should continue in the following areas:
 - Alternative Input Techniques
 - Flight Certified Compiler